PARAMETRIC STRUCTURAL MODELING

User Manual
for Version 1.0.5

written by Clemens Preisinger

March 14, 2014

Contents

1. Introduction 7
   1.1. How to obtain a pro- or pro-student-license 7

2. What's new in version 1.0.5. 8
   2.1. Disclaimer 9

3. Installation 9

4. Quick start 11
   4.1. Basic Example 11
   4.2. Physical Units 12
   4.3. Elements 13
   4.4. Materials 13
   4.5. Cross Sections 14
   4.6. Supports 14
   4.7. Loads 15
   4.8. Model 15
   4.9. Algorithms 15
   4.10. Visualization 16
   4.11. Results 17

5. Quick Component Reference 17
   5.1. License 17
   5.2. Algorithms 17
   5.3. Cross Section 18
   5.4. Ensemble 19
   5.5. Export 20
   5.6. Load 20
   5.7. Material 21
   5.8. Results 21
   5.9. Utilities 22

6. Component Reference 23
   6.1. Ensemble 23
      6.1.1. Activate Element 24
      6.1.2. Assemble Model 24

6.1.3. Connected Parts . . . 25
6.1.4. Disassemble Model . . . 25
6.1.5. Line to Beam . . . 26
6.1.6. Index to Beam . . . 27
6.1.7. Connectivity to Beam . . . 28
6.1.8. Mesh to Shell . . . 28
6.1.9. Disassemble Beam . . . 29
6.1.10. Make Beam-Set . . . 30
6.1.11. Modify Beam . . . 30
6.1.12. Orientate Beam . . . 33
6.1.13. Select Beam . . . 34
6.1.14. Support . . . 35
6.2. Cross Section . . . 37
6.2.1. Box-Profile, Circular Profile, I-Profile and Trapezoid-Profile . . . 38
6.2.2. Shell Cross Section . . . 39
6.2.3. Spring Cross Section . . . 40
6.2.4. Beam Joints . . . 41
6.2.5. Disassemble Cross Section . . . 42
6.2.6. Eccentricitiy on Beam, Eccentricity on Cross Section 42
6.2.7. Cross Section Matcher . . . 43
6.2.8. Cross Section Range Selector . . . 44
6.2.9. Cross Section Selector . . . 44
6.2.10. Generate Cross Section Table . . . 44
6.2.11. Read Cross Section Table from File . . . 46
6.3. Material . . . 47
6.3.1. Material Properties . . . 47
6.3.2. Material Selection . . . 48
6.3.3. Read Material Table from File . . . 48
6.4. Load . . . 49
6.4.1. Point-Load . . . 51
6.4.2. Mesh-Load . . . 51
6.4.3. Line-Load on Element . . . 54
6.4.4. Pretension-Load . . . 54
6.4.5. Temperature-Load . . . 55
6.4.6. Gravity . . . 56
6.4.7. Point-Mass . . . 56
6.4.8. Prescribed displacements . . . 57

6.5. Algorithms . . . 58
6.5.1. Analyze . . . 58
6.5.2. Analyze Large Deformation . . . 59
6.5.3. Eigen Modes . . . 62
6.5.4. Natural Vibrations . . . 64
6.5.5. Force Flow Finder . . . 64
6.5.6. Tension/Compression Eliminator . . . 68
6.5.7. Optimize Cross Section . . . 69
6.6. Results . . . 73
6.6.1. Deformation-Energy . . . 74
6.6.2. ModelView . . . 74
6.6.3. Nodal Displacements . . . 79
6.6.4. Approximate Principal Strains . . . 80
6.6.5. Reaction Forces . . . 81
6.6.6. Utilization of Elements . . . 81
6.6.7. Beam Displacements . . . 82
6.6.8. BeamView . . . 82
6.6.9. Resultant Section Forces . . . 85
6.6.10. Section Forces . . . 87
6.6.11. Force Flow Lines on Shells . . . 88
6.6.12. Isolines on Shells . . . 90
6.6.13. Principal Force Directions on Shells . . . 91
6.6.14. Principal Moment Lines on Shells . . . 91
6.6.15. Principal Stress Directions on Shells . . . 91
6.6.16. Principal Stress Lines on Shells . . . 92
6.6.17. Shell Forces . . . 93
6.6.18. ShellView . . . 93
6.7. Export . . . 94
6.7.1. Export Model to DStV . . . 94
6.8. Utilities . . . 95
6.8.1. Mesh Breps . . . 95
6.8.2. Nearest Neighbors . . . 97
6.8.3. Multi-dimensional Nearest Neighbors . . . 97
6.8.4. Remove Duplicate Lines . . . 98
6.8.5. Remove Duplicate Points . . . 98
6.8.6. Get Cells from Lines . . . 98
6.8.7. Line-Line Intersection . . . 99
6.8.8. Element Felting . . . 99

6.8.9. Mapper . . . 100
6.8.10. Interpolate Shape . . . 101
6.8.11. Proximity Stitch . . . 102
6.8.12. Simple Stitch . . . 103
6.8.13. Stacked Stitch . . . 103
6.8.14. User Iso-Lines and Stream-Lines . . . 104

7. Trouble shooting 105
7.1. Karamba does not work for unknown reason . . . 105
7.2. "fem.karambaPINVOKE"-exception . . . 106
7.3. The StackedStitch-components renders structures with overlapping diagonals . . . 107
7.4. Karamba does not work after reinstalling Grasshoper . . . 107
7.5. Karamba does not appear nor any of its components seem to be installed . . . 107
7.6. Karamba seems to get stuck while calculating a model . . . 108
7.7. Predefined displacements take no effect . . . 108
7.8. The ModelView-component consistently displays all load cases simultaneously . . . 108
7.9. The View-components do not show rendered meshes (stress, strain,...), supports, etc. . . . 108
7.10. The ModelView-component does not display any tags . . . 108
7.11. Circular cross sections show up as flat stripes when rendered 109
7.12. Icons in "Karamba"-toolbar do not show up . . . 109
7.13. Error messages upon loading definitions saved with outdated Karamba versions . . . 109
7.14. Component in old definition reports a run-time error . . . 109
7.15. The "Optimize Cross Section"-component does not work . . . 109
7.16. The "Optimize Cross Section"-component returns wrong results 110
7.17. Other problems . . . 110

A. Background information 111
A.1. Basic Properties of Materials . . . 111
A.1.1. Material Stiffness . . . 111
A.1.2. Specific Weight . . . 111
A.1.3. Theoretical Background of Stiffness, Stress and Strain 112
A.2. Additional Information on Loads . . . 112
A.3. Tips for Designing Statically Feasible Structures . . . 112

A.4. Hints on Reducing Computation Time 114
A.5. Natural Vibrations and Eigen Modes 115
A.5.1. Eigen-modes in Structural Dynamics 115
A.5.2. Eigen-modes in Stability Analysis 116
A.6. Approach Used for Cross Section Optimization 116

1. Introduction

Karamba is a Finite Element program like many others. However it has advantages over these programs in several important respects: It is easy to use for non-experts, has been tailored to the needs of architects in the early design phase, works interactively and costs slightly less than the rest.

Karamba is fully embedded in the parametric environment of Grasshopper which is a plug-in for the 3d modeling tool Rhinoceros. This makes it easy to combine parameterized geometric models, finite element calculations and optimization algorithms like Octopus or Galapagos.

Besides the free version for non-commercial use only, there exists also a pro-version of Karamba for commercial use and a trial-version. Table 1 lists their main features.

Table 1: Variants of Karamba

Karamba version	beam elements	shell elements	other features
free	unlimited	≤ 50	limited
trial	≤ 20	≤ 50	unlimited
pro	unlimited	unlimited	unlimited
pro-student	unlimited	unlimited	unlimited

1.1. How to obtain a pro- or pro-student-license

The commercial license ("pro") without time limitation costs € 30 for students and € 990 for businesses. A pro-license with one year validity comes at € 390. The pro-student version is for non-commercial use only. In case you plan to use Karamba for university courses please contact us for special conditions.

One single static license can be installed on two computers. A license includes all updates of the current main version (i.e. currently 1.x.x.).

For commercial use a network license option exists. Being based on the Zoo 5.0 license server of Mc Neel it comes at the same price as a static license. Versions of one year and unlimited validity can be purchased.

In order to obtain the pro-version download Karamba from either `http://www.karamba3d.com` or `http://www.food4rhino.com/project/karamba` and install the trial-version. Then use the Karmba-license component (right-click on the icon) to generate a "machine.id"-file on the computers where you wish to run Karamba on. Send these files to `info@karamba3d.com` and buy a license

at `http://www.karamba3d.com/downloads/`. In return you will get a "license.lic"-file which turns your Karamba trial into a Karamba pro or pro-student version. Start Rhino as an administrator, right-click on the Karamba-license component and select "Load license file".

When purchasing a student version either attach a scan of your student ID or send the e-mail from your university account. More information regarding the pro-version can be obtained through the "License"-component (see fig. 1).

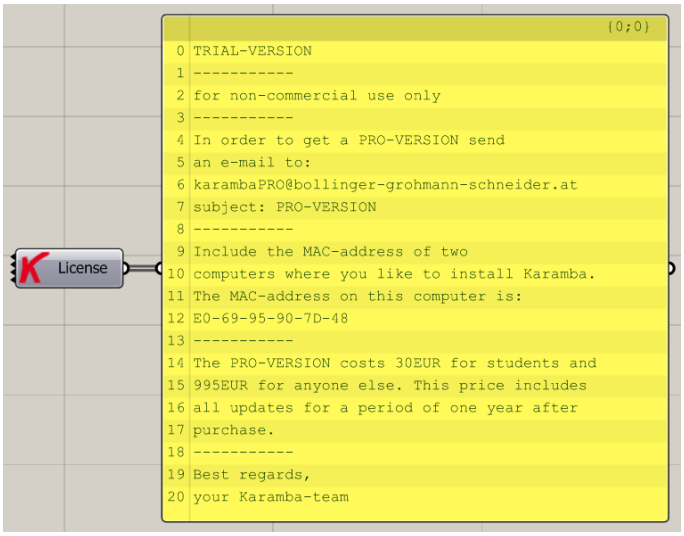

License
{0;0}
0 TRIAL-VERSION
1 ----------
2 for non-commercial use only
3 ----------
4 In order to get a PRO-VERSION send
5 an e-mail to:
6 karambaPRO@bollinger-grohmann-schneider.at
7 subject: PRO-VERSION
8 ----------
9 Include the MAC-address of two
10 computers where you like to install Karamba.
11 The MAC-address on this computer is:
12 E0-69-95-90-7D-48
13 ----------
14 The PRO-VERSION costs 30EUR for students and
15 995EUR for anyone else. This price includes
16 all updates for a period of one year after
17 purchase.
18 ----------
19 Best regards,
20 your Karamba-team

Figure 1: The "License"-component

Those parts of this manual that apply to the pro/trial-version only, are either blue or have blue section headings.

2. What's new in version 1.0.5.

- A simplified installation procedure.
- Iso- and stream-lines with user supplied values and vectors at mesh-vertices (unlimited mesh-size also in the free-version of Karamba).
- Benchmark examples with known results from literature for the evaluation of calculation accuracy.
- Cross section optimization comprises now also shells.
- Shell elements can be given individual thicknesses.

- Line-like shell results can be plotted on the deformed mesh.
- Faster model assembly and generation of mesh-loads as compared to the previous version of Karamba.
- A component for the retrieval of distributed cross section forces for shells.
- Principal moment- and normal force directions on shells.
- Separation of graphical user interface and underlying functionality of most Karamba components. This makes it easier to use Karamba in scripts.

These bugs got fixed:

- The layer-position of shell results was corrected.
- A problem with infinite loops in the calculation of flow-lines was resolved.
- Snapping nodes of elements do not cause errors and problems with mesh-loads any more.
- A bug that led to deviations in the display of principal stresses from the correct direction was removed.

Due updating to .NET Framework 4.0 Karamba 1.0.5 does not run on Rhinoceros 4 any more – sorry.

2.1. Disclaimer

Although being tested thoroughly Karamba probably contains errors – therefore no guarantee can be given that Karamba computes correct results. Use of Karamba is entirely at your own risk. Please read the license agreement that comes with Karamba in case of further questions.

3. Installation

These are the prerequisites for installing Karamba:

- **Rhino 5.0**

- **Grasshopper**
  version 1.0.5 of Karamba was tested on GH 0.9.0056.

In case you do not possess Rhino download a fully featured, free trial version from `http://www.rhino3d.com/download.html`. Grasshopper which is free can be found at `http://www.grasshopper3d.com/`.

Karamba comes as 32-bit or 64-bit application. Select the version according to the bitness of the Rhino version you want to work with. By default 'Rhinoceros 5' (this is the 32-bit version) and 'Rhinoceros 5 (64-bit)' get installed.

The installation procedure lets you set the physical units used for calculation. By default Karamba assumes input to be in SI units (e.g. meters for point coordinates). You can switch to Imperial units either on installation or later on by editing the "karamba.ini" file. Coordinates will then be interpreted to be in feet, force in "kips", material strength in "ksi" and so on.

For installation invoke "KarambaSetup.msi". It automatically installs "karamba.dll" and "libiomp5md.dll" to "C:/Windows/" and all other files to the "Components Folder". Its location can be easily found by selecting "File/Special Folders/Component Folders" from the Grasshopper menu. Besides other things a folder named "Karamba" will be created there, containing the license agreement, a readme-file, pre-fabricated cross section and material tables and the configuration file "karamba.ini". The config-file contains general program settings and can be edited with any text editor. It contains key - value pairs and is pretty self descriptive.

If all goes well you will notice upon starting Grasshopper that there is a new category called Karamba on the component panel. It consists of ten subsections (see figure 2). In case you do not see any icons select "Draw All Components" in Grasshoppers "View"-menu. The installation can be tested by placing a Karamba "License"-component on the canvas: it should not issue a warning or error. If not see section 7.2 for how to solve that issue. On Apple machines make sure to have Microsofts .NET Framework 4.0 installed.

These are the subsections which show up in the Karamba category:

- "License": The 'License'-component contained in here delivers information regarding the current type of license and how to get a pro-version of Karamba (see fig. 1).

- "Params": containers for Karamba objects like beams, loads, models,...

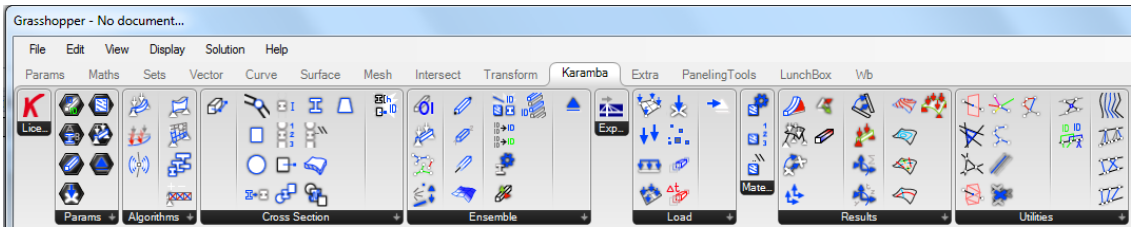


Figure 2: Category "Karamba" on the component panel

- "Algorithms": components for analyzing the structural model
- "Cross Section": contains components to create and select cross sections for elements.
- "Ensemble": lets you create models
- "Export": for export of Karamba-models to RStab or Robot via DStV-file.
- "Load": components for applying external forces
- "Materials": components for the definition of material properties
- "Results": for the retrieval of calculation results
- "Utils": contains some extra geometric functionality that makes it easier to handle and optimize models.

The colors of Karamba's icons have a special meaning: black or white designates the entity or entities on which a component acts. Products of components get referenced by a blue symbol.

Basic knowledge of Rhino and Grasshopper is needed to understand this guide. It is recommended to download the Grasshopper Primer from the Grasshopper web-site for introductory material on Grasshopper.

4. Quick start

4.1. Basic Example

Creating a statical model in Karamba consists of six basic steps[1] – see fig. 3:

[1] This step-by-step procedure was devised and formulated by Justin Diles

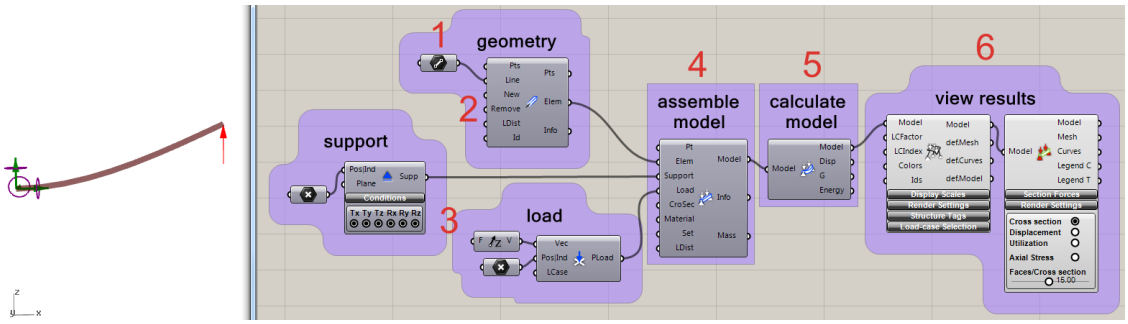


Figure 3: Basic example of a statical model in Karamba

1. **Create** wire-frame, point geometry or meshes for the structural model with Rhino or GH.
2. **Convert** wire-frame or point geometry to Karamba beams, meshes to shells.
3. **Define** which points are supports and which receive loads.
4. **Assemble** the Karamba structural model with points, elements, supports and loads. Optional: Define custom cross sections and materials and add them as well. They reference elements either by index or user defined element identifiers.
5. **Analyze** the Karamba structural model.
6. **View** the analyzed model with the "ModelView"-component. Deflections can be scaled, multiple load cases can be viewed together or separately. The "BeamView"- and "ShellView"-components (not shown in fig 3) can be used to generate mesh representations of stresses, level of material utilization,....

Karamba is intended to provide an intuitive approach to statical modeling. All its components come with extensive help-tags and there are lots of examples on the web (see `http://www.karamba3d.com/examples` or `http://www.grasshopper3d.com/group/karamba/page/example-files`) and in the installation folder which can be easily customized according to ones own needs.

4.2. Physical Units

On installing Karamba one can specify the family of physical units to be used for input and results. The default option is metric (e.g. meters, centimeters,

degree Celsius, Newtons,...) but Karamba can also deal with Imperial units (e.g. feet, inch, degree Fahrenheit, kiloponds,...).

The set of units to be used can be changed any time by editing the "karamba.ini" file.

Depending on the family of units Karamba interprets geometric input either as meters or feet. The kind of physical units that components expect to receive shows up in the tool-tip which appears when the mouse pointer hovers over an input-plug.

Changing the type of physical units during the creation of a GH definition may give rise to problems: The help text of Grasshopper components does not change dynamically. Switching from SI to imperial units leaves the help text of those components already placed on the canvas unaltered. The interpretation of the input values however changes. Opening a GH definition with Karamba versions with differently set physical units entails the same problem.

4.3. Elements

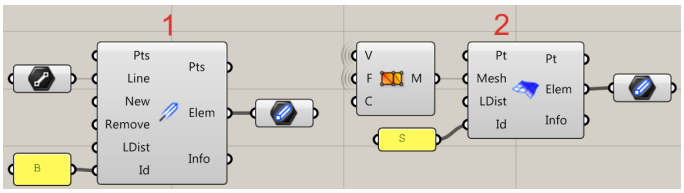


Figure 4: Components for creating beam- (1) and shell-elements (2)

Beam- and shell-elements can be generated based on lines and meshes. Give an "id" name for later reference (e.g. "B" and "S" in fig. 4). This allows non-default cross sections and materials to be assigned to them. Element names need not be unique. Karamba provides truss-, beam- and shell-elements. All geometric input is assumed to be in meters when you use SI-units and feet in case of Imperial units.

4.4. Materials

Materials can be defined by setting their mechanic properties. Alternatively materials can be selected from a library of predefined materials that comes with Karamba. The "Elem Id" input-plug specifies the names of the elements to which the material shall be attached. Leaving "Elem Id" empty defines

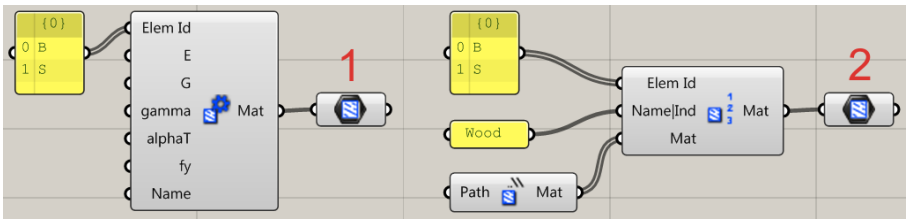


Figure 5: left: definition of a custom material (1). Right: selection of a material from the material library (2)

the material for all elements. Steel is the default for elements if nothing is given.

4.5. Cross Sections

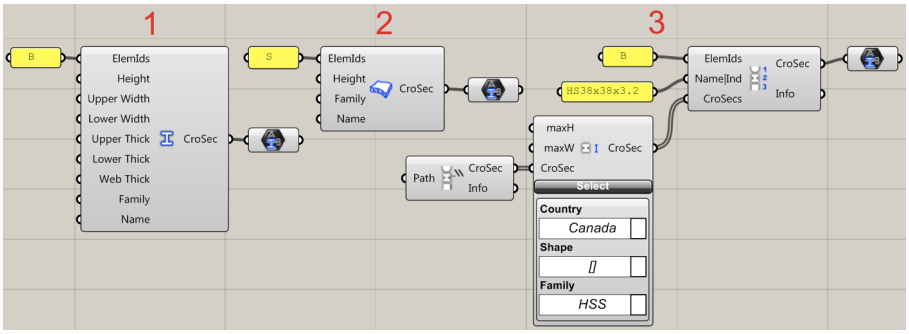


Figure 6: left: definition of a beam cross section (1); Middle: definition of a shell cross section (2); Right: selection of a cross section from the cross section library (3)

Arbitrary I-, hollow box, filled trapezoid and hollow circular cross sections can be defined for beams, an element-wise variable height for shells. Cross sections get attached to elements via element names (”B” and ”S” in fig. 6). Alternatively cross sections can be chosen from a library.

4.6. Supports

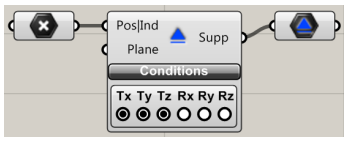


Figure 7: Component for creating supports.

Supports suppress translations or rotations at nodes. An activated button appears black and means either zero translation (T) in the direction of the global x-, y- or z-axis or zero rotation (R) about the corresponding global axis. A node index or position can be used to specify the location of a support. Supply a plane as input for specifying locally oriented support conditions.

4.7. Loads

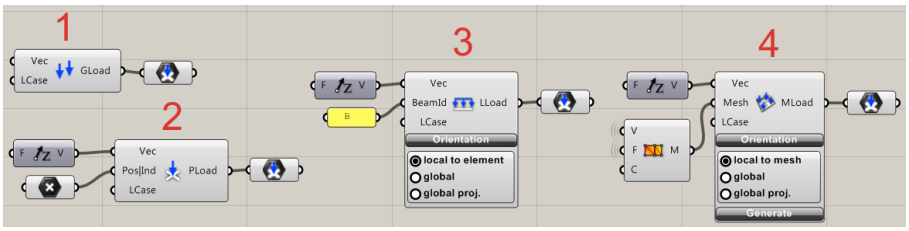


Figure 8: definitions of gravity load (1), point load (2), uniformly distributed load on a beam (3) and distributed load on a mesh (4)

Fig. 8 shows examples for defining loads on parts of a structural model. The input-plug "LCase" can be used to set the number of the load case in which the load acts. This enables different load scenarios (e.g. wind from different directions) to be created. Gravity loads (1) act on the whole structure. The location of point loads (2) can be specified by node index or position. Beam loads (3) act on elements given by element identifiers. Distributed loads on arbitrary meshes (4) get reduced to approximately statically equivalent node and beam loads. The direction of gravity and point-loads is given by a vector in input-plug "Vec" and refers to the global coordinate system. The direction vector of beam- and mesh-loads can be specified relative to the global or local (relating to the element or mesh) coordinate system.

4.8. Model

After defining Elements, supports, loads and optionally cross sections and materials the statical model can be assembled (see fig. 9). Elements get rigidly connected in case they attach to the same node.

4.9. Algorithms

Karamba offers several different ways of evaluating a structural model. The "Analyze" component calculates the response of a model under external

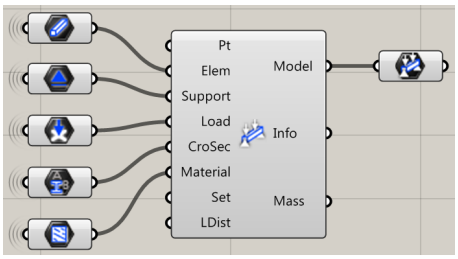


Figure 9: The model gets assembled from the generated structural information.

Figure 10: The model can be evaluated in several ways. Left: analysis of structural response under loads; Right: calculation of eigen-modes.

loads. The numerical evaluation options comprise eigen-modes, natural vibration modes, large deflections, evolutionary structural optimization, cross section optimization and iterative elimination of tension or compression elements. For each calculation option exists a corresponding component which takes a model as input, calculates it and adds the results to the model data.

4.10. Visualization

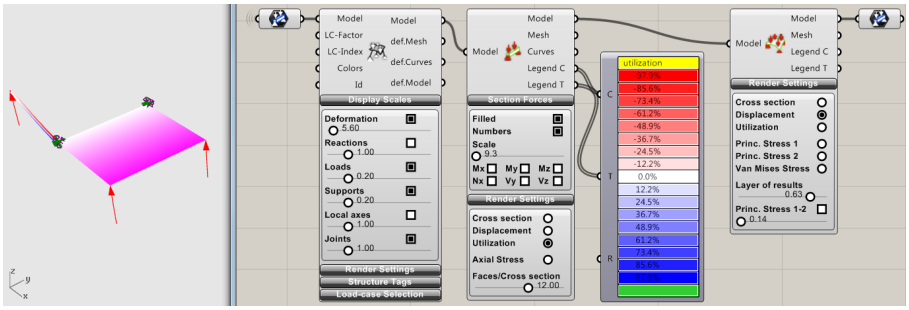


Figure 11: There are three components for visualizing the model: "ModelView", "BeamView" and "ShellView"

Karamba comes with three components for visualizing the structural model (see fig. 11):

1. "ModelView": Sets the basic visualization properties like scaling factor of displacements, sizes of symbols, number of displayed load case,...

2. "BeamView": visualizes beams

3. "ShellView": visualizes shells

Each of these components contains submenus which can be unfolded by clicking on the black caption bar. The numerical range of sliders can be set by double clicking on their black knob. Visualization properties stick to the model and stay valid until they get overruled by another downstream visualization component.

4.11. Results

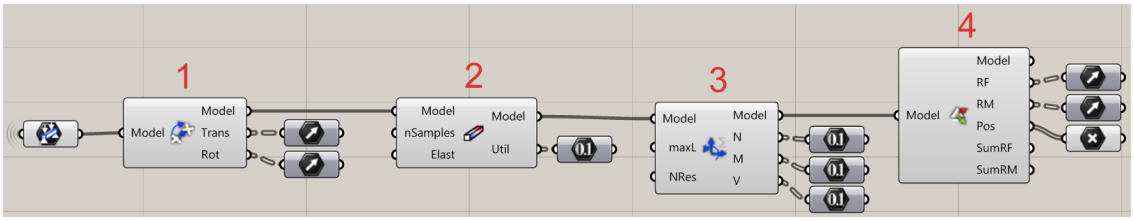


Figure 12: Retrieval of numerical results: nodal displacements (1), level of material utilization (2), resultant cross section forces (3) and reaction forces (4).

Structural response properties can be used to inform the model and e.g. optimize it. Fig. 12 shows some of the available options.

5. Quick Component Reference

5.1. License

License: Returns the program version, license information and can be used to manage the license file.

5.2. Algorithms

Analyze: Calculates the deflections of a given model.

Analyze Large Deformation: Does incremental geometrically nonlinear analysis for loads in load case zero.

Natural Vibrations: Calculates the natural vibrations of the given model.

Eigen Modes: Calculates the eigen-modes of the given model according to the special eigenvalue problem.

Force Flow Finder: Optimizes a structure by placing beams along the route which external forces take through the structure.

Optimize Cross Section: Selects optimum cross sections for beams and trusses in the model.

Tension/Compression Eliminator: Removes beams or trusses under axial tension or compression. By default compression members will be removed.

5.3. Cross Section

Box-Profile: Creates rectangular, trapezoid and triangular hollow cross sections.

Circular Hollow Profile: Creates circular hollow cross sections.

I-Profile: Creates I-shaped cross sections.

Shell Cross Section: Lets you set the height of a shell cross section.

Spring-Cross Section: Defines the spring stiffness of an element.

Trapezoid-Profile: Creates filled rectangular, trapezoid and triangular cross sections.

Beam-Joints: Adds hinges at the end-points of beams.

Disassemble Cross Section: Retrieves properties of a cross section.

Eccentricity on Beam: Sets the eccentricity of a cross section relative to the element axis in global coordinates.

Eccentricity on Cross Section: Sets the eccentricity of a cross section relative to the element axis in local beam coordinates.

Cross Section Matcher: Returns for a cross section the best fitting cross section contained in a given list. The matched cross section is equal or better in all mechanical aspects at minimum weight.

Cross Section Range Selector: Lets you select cross sections by country, shape, family or maximum depth or width.

Generate Cross Section Table: Converts a list of cross sections into a string which can be streamed as a csv-file and used as a cross section table.

Cross Section Selector: Lets you select cross sections by name, regular expression or index from a list of cross sections.

Read Cross Section Table from File: Reads cross section data from a csv-file.

5.4. Ensemble

Activate Model: Activates the elements of a model according to the activation list. Uses soft kill approach for inactive elements.

Assemble Model: Creates a finite element model by collecting given entities (points, beams, shells, supports, loads, cross sections, materials,...).

Connected Parts: Returns groups of interconnected lines of the model.

Disassemble Model: Decomposes a model into its components.

Connectivity to Beam: Creates beams with default properties from given connectivity diagram.

Index to Beam: Creates beams with default properties from given node indexes.

Line to Beam: Creates beams with default properties from given lines. Lines that meet at a common point are by default rigidly connected with each other. Karamba assumes input to be in meter or feet.

Mesh to Shell: Creates shells with default properties from given meshes. Quad faces are split to triangles.

Disassemble Beam: Decomposes beams into their components.

Make Beam-Set: Puts beams designated by their beam identifier into a group.

Modify Beam: Modifies beam properties like total section height, wall thickness of cross section, activation state and whether it has bending stiffness or not.

Orientate Beam: Sets the local Z-axis of beams according to a given vector and adds a rotation angle DAlpha $[deg]$ about the longitudinal axis. Flips beam direction according to a given x-vector.

Select Beam: Selects beams according to a given identifier and puts all incoming beams in two groups: selected or rejected. The identifier may be the element index, name or a regular expression.

Support: Creates supports at nodes of given node-indexes or node-coordinates. Lets you select translations/rotations which should be zero and the support orientation with respect to the global coordinate system.

5.5. Export

Export Model to RStab: Exports a model to RStab5, RStab6, RStab7 or Robot by creating a DStV-file.

5.6. Load

Dissemble Mesh Load: Splits a mesh-load into corresponding line- and point-loads.

Gravity: Creates gravity from a specified direction vector for given load-cases.

Line-Load on Element: Creates a uniformly distributed load on a beam.

Mesh-Load: Creates approximately equivalent point- and line-loads from a surface load on a mesh.

Point-Load: Creates point loads at points of given index or position.

Point-Mass: Attaches a point mass to a node of given index or position. Does not result in additional weight, only translational inertia.

Pretension-Load: Sets pretension loads on beams: The elements get deformed and then built into the structure.

Temperature-Load: Imposes a temperature difference on an element with respect to its initial temperature at construction.

Prescribed Displacement: Prescribes displacements at nodes of given node-indexes or node-coordinates. Select translations or rotations which should be prescribed. For load-cases with no displacements prescribed this will create a support.

5.7. Material

Material Properties: Sets the characteristic parameters of a material.

Material Selection: Lets you select a material by name, regular expression or index from a list of materials.

Read Material Table from File: Reads a list of materials from a table given in csv-format.

5.8. Results

Deformation-Energy: Retrieves deformation energies of the elements of the model.

Model View: Lets you inspect the general properties of the model.

Nodal Displacements: Returns nodal displacements: translations/rotations in global x-, y-, and z-direction; rotations about global x-, y- and z-axis.

Principal Strains: Approximates the principal strain directions from the model deformation at arbitrary points.

Reaction Forces: Returns reaction forces and moments at supports.

Utilization of Elements: Returns the utilization of elements for each load case. The utilization of beams is calculated according to EC3 (see section A.6), for shells the maximum Van Mises Stress serves as the basis. "1" means 100%.

Beam Displacements: Returns displacements along elements: translations/rotations in global x-, y-, and z-direction; rotations about global x-, y- and z-axis.

Beam View: Lets you inspect beam properties: section forces, cross sections, displacement, utilization and stresses. Is to be plugged into the definition after the ModelView-component.

Section Forces: Retrieves section forces along beams and trusses.

Resultant Section Forces: Retrieves resultant section forces of beams.

Force Flow Lines on Shells: Computes flow lines for forces in given direction at user defined positions.

Isolines on shells: Creates lines that connect points of same value for selected shell results (e.g. principal stresses, displacement, utilization, cross section thickness) at user defined positions. Also returns values and can thus be used for probing the shell state.

Principal Moment Lines on Shells: Lets you retrieve the direction and absolute value principal shell cross section forces and moments in the center of each shell face.

Principal Moment Lines on Shells: Returns the principal moment lines that originate from user defined points on shells.

Principal Stress Directions on Shells: Outputs the principal stress directions in the center of each shell element.

Principal Stress Lines on Shells: Returns the principal stress lines that originate from user defined points on shells.

Principal Stress Directions on Shells: Outputs the values of first and second principal normal forces and moments in the center of each shell element.

Shell View: Lets you inspect shell properties: displacement, utilization, principal stresses and Van Mises stress. Is to be plugged into the definition after the ModelView-component.

5.9. Utilities

Detect Collisions: Counts the number of intersections between the model and a given mesh.

Get Cells from Lines: Creates closed cells from a graph and vertices on a user supplied plane.

Line-Line Intersection: Intersects given lines and returns resulting end-points and pieces.

Mesh Breps: Takes multiple Breps and generates a unified mesh from them. The algorithm takes account of common edges and predefined points. This lets one define positions for supports or point-loads on shells.

Nearest Neighbors: Connects each node of one set to a given number of nearest neighbor nodes or neighbors within a specified distance of another set.

Multi-dimensional Nearest Neighbors: Performs a multidimensional nearest neighbor search on a two sets of vectors.

Remove Duplicate Lines: Eliminates identical lines.

Remove Duplicate Points: Eliminates identical points.

Element Felting: Felts elements of a model by connecting them at their mutual closest points.

Mapper: Applies mappings (like Simple Stitch) to a model.

Interpolate Shapes: Interpolates between a base geometry (0.0) and given shape(s) (1.0).

Proximity Stitch: Connects beam sets by a preset number of elements whose maximum inclination can be controlled via min/max offset-limits from their starting point.

Simple Stitch: Connects beam sets by a preset number of elements.

Stacked Stitch: Connects beam sets by a preset number of elements that do not intersect each other.

User Iso-Lines: Creates iso-lines on a model based on user supplied nodal values.

User Stream-Lines: Creates stream-lines on a model based on user supplied vectors at the nodes.

6. Component Reference

6.1. Ensemble

The subsection "Ensemble" of Karamba contains components for handling the basic aspects of a statical model.

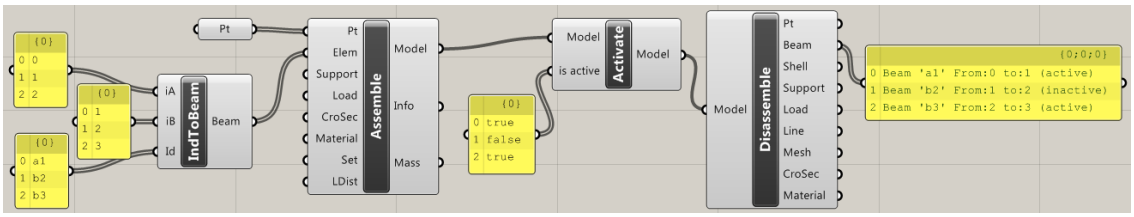


Figure 13: Setting the activation state of all elements of a model with a list of boolean values.

6.1.1. Activate Element

The activation state of an element can be controlled with the Activate Element-component (see fig. 13). This component expects a model and a list of boolean values as input. The list of true/false values will be mapped to the activation status of the elements in the model. "true" corresponds to "active", "false" to inactive. Section 6.5.5 shows, how the Activate Element-component enables one to view the solution history of the iterative FindForcePath-algorithm.

Karamba sets elements inactive by giving them a very weak material with zero weight.

6.1.2. Assemble Model

In order to calculate the behavior of a real world structure one needs to define its geometry, loads and supports. The component "Assemble" gathers all the necessary information and creates a statical model from it (see figure 14).

In case that some beams were defined by node indexes then these will refer to the list of points given at the "Pt" input-plug.

The input-plug "LDist" can be used to define the distance of points below which they will be merged to one. This helps in dealing with inaccurate geometry. Giving a negative value to "LDist" allows two separate nodes in a model to reside on the same spot. This lets you define zero length elements such as springs connecting the two halves of a scissor structure.

The output-plug "Mass" renders the mass of the structure in kilogram. When being plugged into a panel the model prints basic information about itself: number of nodes, elements, and so on. At the end of the list the characteristic length of the model is given which is calculated as the distance between opposing corners of its bounding box.

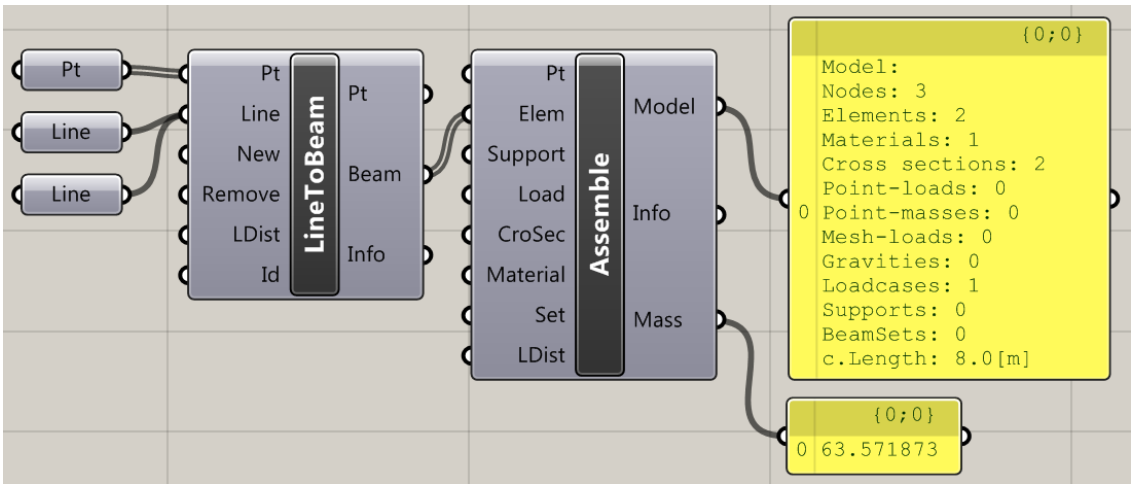


Figure 14: The Assemble-component gathers data and creates a model from it.

6.1.3. Connected Parts

When creating a model based on unprecise geometry or using a generative process, elements may not be connected to each other as desired. The Connected Parts-component (see fig. 15) takes a model as input and determines its connected parts. It considers beams and trusses only. Connected groups get listed in a data tree in descending order of group size.

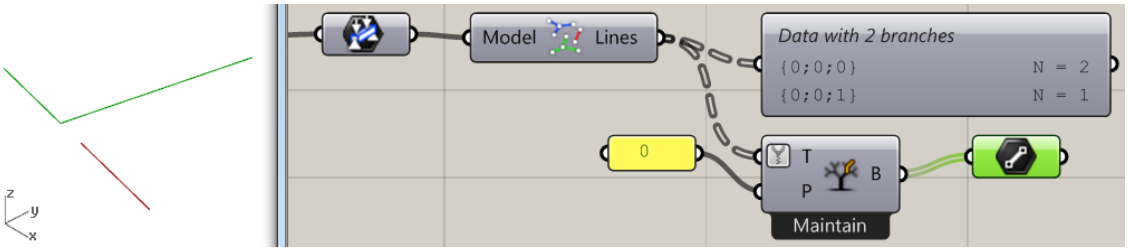


Figure 15: The Connected Parts-component groups beams into sets of elements that have at least on node in common each.

6.1.4. Disassemble Model

It is sometimes necessary to pull apart existing models in order to reassemble them in different configurations. The DisassembleModel-component can be used for decomposing a statical model into its components (see figure 16). Loads, supports and elements reference the nodes they connect to by their node-index – regardless whether they were initially defined using coordinates

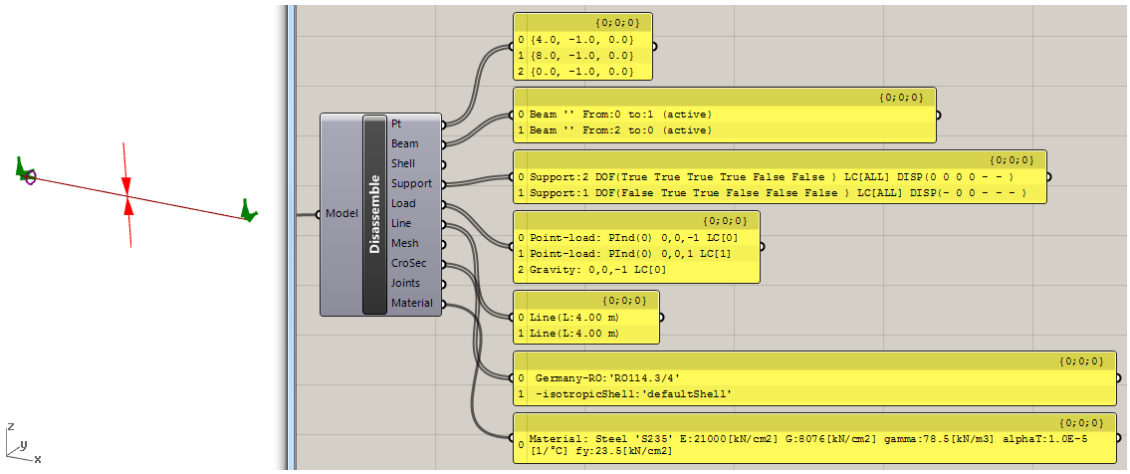


Figure 16: Model is decomposed into its components.

or node-indexes. This can be used to change the geometry of a model without altering its topology: plug a list of modified Grasshopper points into the "Pt"-plug of an Assemble-component along with the objects you get from DisassembleModel and see what happens.

6.1.5. Line to Beam

Figure 17 shows how Karambas LineToBeam-component takes two lines as input, finds out how they connect and outputs beams as well as a set of unique points which are their end-points. Points count as identical if their distance is less than that given in "LDist". The default value is $0.005[m]$. The LineToBeam-component accepts only straight lines as geometric input. Therefore poly-lines and the like need to be exploded into segments first.

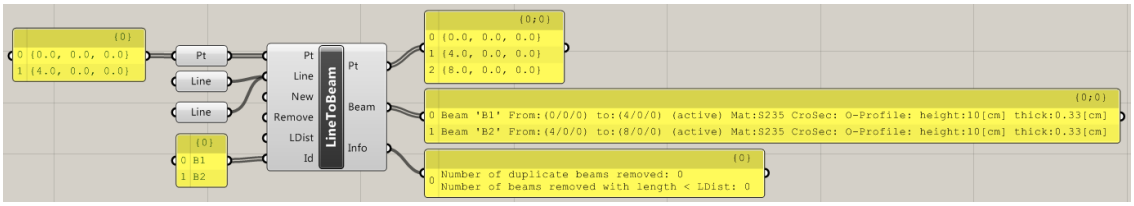


Figure 17: The LineToBeam-component that turns two lines into beams

All coordinates are in meters. In order to be of immediate use beams come with a number of default values (see string-output in figure 17): "active" means that a beam will be included in the statical model. The default cross section is a circular hollow profile of diameter $11.4[cm]$ with a wall-thickness

of $0.4[cm]$. The default material is steel of grade "S235".

Lines that fall below the length limit given by "LDist" get automatically removed. Unless lines get removed there is a one to one correspondence between the list of input lines and output beams. The order in which points appear in the output node-list is random by default. However it is sometimes advantageous to identify certain points by their list index in order to put loads on them or to define supports. This can be achieved by feeding a list of coordinates into the "Points"-plug. They will be placed at the beginning of the output nodes-list. So in order that the end-points of the structure in figure 17 have index 0 and 1 it is necessary to input a list of points with coordinates (0/0/0) and (8/0/0).

There are four more input plugs on the LineToBeam-component:

- New: If this plug has the value "False" only those lines will be added to the structure that start and end at one of the points given in the input points-list.
- Remove: If this option has the value "True" the LineToBeam-component checks for lines that lie on each other and merges such duplicates into one. This prevents an error that is hard to detect by visual inspection alone: Two lines on the same spot mean double member stiffness in the statical model.
- LDist: sets the limit distance for two points to be merged into one. Lines of length less than that value will be discarded. The default value is $5[mm]$.
- Id: takes a list of strings as identifiers for beams. The default value is an empty string. Each beam has a name by default: its zero based index in the model. Identifiers provide a useful means to group the beams in order to modify or display them.

Beams that meet at a common point are by default connected rigidly in the statical model like they were welded together. See section 6.2.4 on how to define joints at the end of beams. The "Info" output-plug informs about the number of removed nodes and beams.

6.1.6. Index to Beam

Sometimes the initial geometry is already given as a set of points and two lists of node-indexes with one entry for each start- and end-point of beams re-

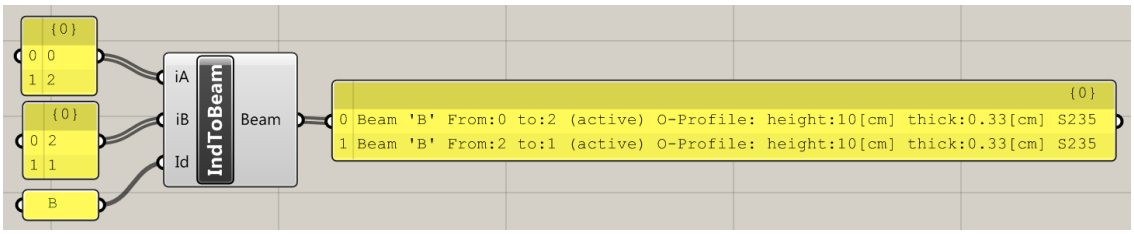


Figure 18: The IndexToBeam-component lets you directly define the connectivity information of beams

spectively. In such a case it would be cumbersome to convert this information into geometric entities only for feeding it into the LineToBeam-component which reverses the previous step. The IndexToBeam-component (see figure 18) accepts a pair of lists of node-indexes and produces beams with default properties from it. This speeds up model generation considerably for there is no need to compare nodes for coincident coordinates.

The IndToBeam-component makes it possible to define elements with zero length. This proves useful in case you want to connect elements that touch each other but should not be rigidly connected (think of a scissor – see section 6.2.3 about springs).

6.1.7. Connectivity to Beam

In Grasshopper meshing algorithms can generate topological connectivity diagrams. With the help of the ConToBeam-component these may be directly converted to beam-structures (see figure 19).

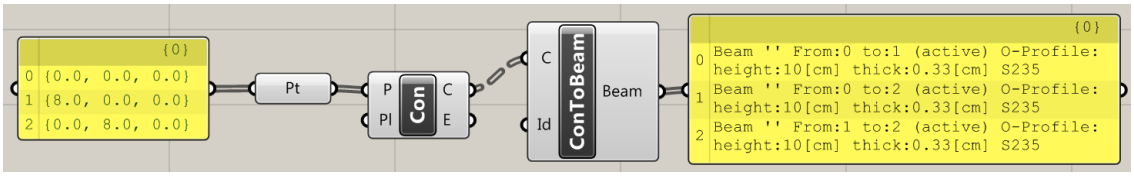


Figure 19: The ConToBeam-component turns connectivity diagrams into sets of beams

6.1.8. Mesh to Shell

The "MeshToShell" component takes a triangle or quad mesh and turns it into a group of shell elements (see fig. 20). Quads get automatically decomposed to triangles. Each patch of shells can be given an identifier for later reference when attaching custom material or cross section properties.

Shell patches are rigidly connected when their nodes lie at a distance less then that given in "LDist". The "Pt" input serves the same purpose as in the "LineToBeam"-component – see sec. 6.1.5. By default shells have a thickness of $1[cm]$ and steel as their material.

The shell elements used in Karamba resemble the TRIC-element devised by Argyris and coworkers (see [1], [2] for details). They are facetted (i.e. flat) elements with constant strain in each layer. Karamba neglects transverse shear deformation.

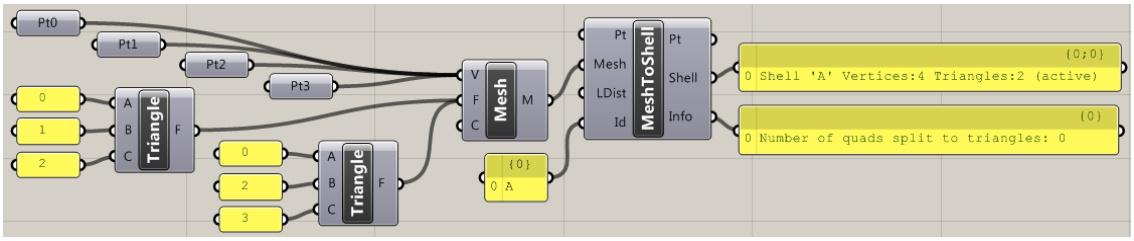


Figure 20: The MeshToShell-component turns meshes into shells

6.1.9. Disassemble Beam

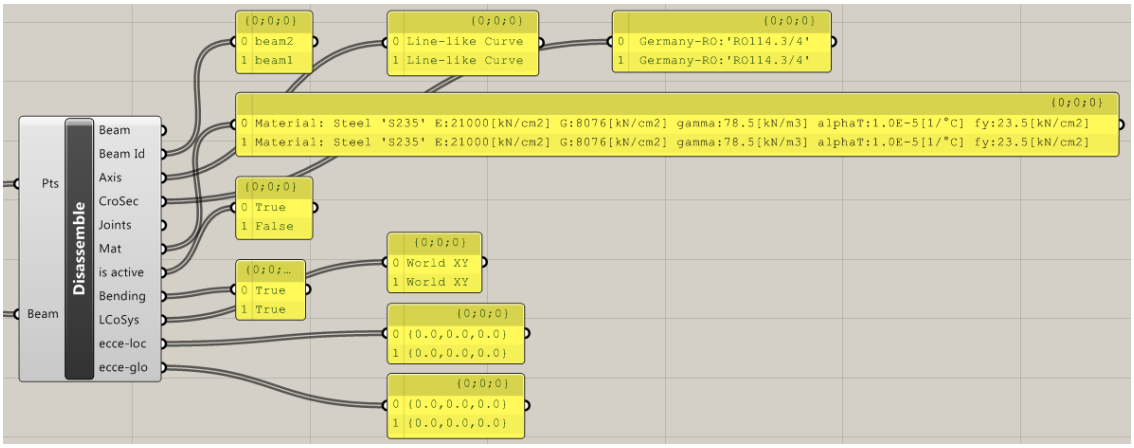


Figure 21: A beam decomposed into its individual parts.

When interested in the information contained in a beam component feed it into the "DisassembleBeam"-component (see fig. 21). It is necessary to provide the list of points that correspond to the list of nodes of the model. This is due to the fact that beams can reference their nodes via index.

6.1.10. Make Beam-Set

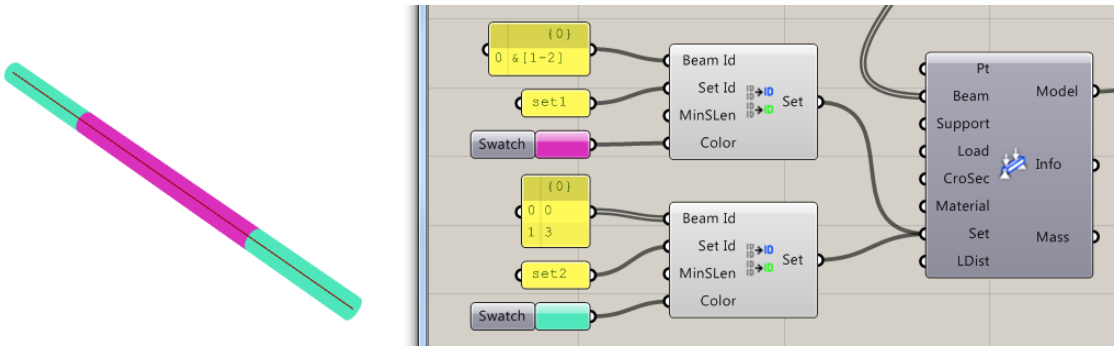


Figure 22: Beam-sets can be used to group beams.

The "Make Beam-Set"-component provides a practical way for grouping different elements under one identifier (see fig. 22). Beam-sets need not be disjoint. The "Beam Id" plug expects a list of strings with beam-identifiers, beam indexes, other beam-set-identifiers or a regular expression. Regular expressions have "&" as their first character by definition. "Set Id" expects a string which serves as identifier of the new set of beams.

The group of beams defined by a set can be used for defining geometric mappings. In this context a beam-set represents a polygon of straight segments. The order of the elements in the set is defined by the order in which they were entered into the set. Such polygons can be split at an arbitrary position (see e.g. section 6.8.12). "MinSLen" (minimum segment length) lets you set the minimum length which may result from such a split. In case of potentially smaller segments the intersection point snaps to its nearest neighbor.

In order to group a structure visually, beam-sets can be given different colors. These colors show when "Cross section" is enabled in the BeamViews "Render Settings" (see section 6.6.8).

The identifier of a beam-set can be used anywhere instead of a beam identifier. In order to be registered with the model, beam-sets need to be fed into the "Set" input-plug of the Assemble-component.

6.1.11. Modify Beam

By default Karamba assumes the cross-section of beams to be steel tubes with a diameter of $14.3[cm]$ and a wall-thickness of $0.4[cm]$. When two beams meet they are rigidly connected like they were welded together. Use

the ModifyBeam-component to set the beam properties according to your choice. Figure 23 shows how this can be done by inserting it in front of the Assemble-component. By default the ModifyBeam-component leaves all incoming beams unchanged. Negative values for input properties take no effect. Several ModifyBeam-components may act consecutively on the same beam.

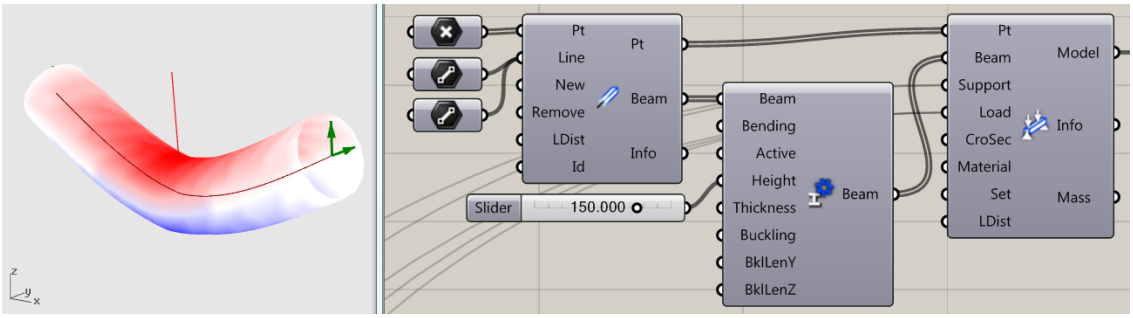


Figure 23: Modification of the default beam properties.

Bending stiffness

Beams resist normal force and bending. Setting the "Bending"-plug of the ModifyBeam-component to false disables bending stiffness and turns the corresponding beam into a truss. There exist reasons that motivate such a step:

- Connections between beams that reliably transfer bending and normal force are commonly more expensive than those that carry normal force only. The design of connections heavily depends on the kind of material used: rigid bending connections in wood are harder to achieve than in steel. Yet rigid connections add stiffness to a structure and reduce its deflection. Therefore you are always on the safe side if you use truss elements instead of beams.
- For slender beams i.e. beams with small diameter compared to their length the effect of bending stiffness is negligible compared to axial stiffness. Just think of a thin wire that is easy to bend but hard to tear by pulling.
- Abandoning bending stiffness reduces computation time by more than half for each node with only trusses attached.
- Karamba bases deflection calculations on the initial, undeformed geometry. Some structures like ropes are form-active. This means that when

a rope spans between two points the deformed geometry together with the axial forces in the rope provide for equilibrium. This effect is not taken into account in Karamba. In Karamba only the bending stiffness of the rope (which is very small) keeps it from deflecting indefinitely. One way to circumvent this lies in using a truss instead of a beam-element. The second possibility would be to reduce the specific weight of the rope to zero (see further below). The third possibility would be to start from a slightly deformed rope geometry and apply the external loads in small steps where the initial geometry of each step results from the deformed geometry of the previous one (see section 6.5.2).

Trusses only take axial forces. Therefore they do not prevent the nodes they are connected to from rotating. In case that only trusses attach to a node, Karamba automatically removes its rotational degrees of freedom. Otherwise the node could freely rotate which is a problem in static calculations. As soon as one beam connects to a node the node has rotational degrees of freedom. Bear this in mind when the Analysis-component turns red and reports a kinematic system. Transferring only axial forces means that a truss reduces a nodes movability in one direction. A node that is not attached to a support has three translational degrees of freedom. Thus there must be three truss elements that do not lie in one plane for a node to be fixed in space.

Activation status of beams

When set to false this option excludes the corresponding beam from further calculations until it is reset to true. See section 6.1.1 for an alternative way of setting a beams activation state.

Height and wall-thickness of cross-sections

Height – which in case of circular tubes is equivalent to the outer diameter D – and wall-thickness of a cross-section determine a beams axial and bending stiffness. Karamba expects both input values to be given in centimeter. The cross-section area is linear in both diameter and thickness whereas the moment of inertia grows linearly with thickness and depends on D^3 in case of full sections and on D^2 in case of hollow sections. So in case of insufficient bending stiffness it is much more effective to increase a beams height (or diameter) than increasing its wall thickness.

Buckling property for cross section optimization

Buckling can be turned off for cross section optimization. This lets you simulate pre-tensioned, slender elements without having to really pretension

them. The necessary pretension force is the negative value of the largest compressive axial normal force of all load cases.

Buckling length in local beam directions

For doing cross section optimization it is necessary to know a beams buckling length. Karamba approximates it using the algorithm described in section 6.5.7. For cases of system buckling this approximation does not lie on the safe side. The input-plugs "BklLenY" and "BklLenZ" allow to specify the buckling length of a beam for its local Y- and Z- axis respectively. When specified, these values override those from the buckling length calculation of Karamba.

6.1.12. Orientate Beam

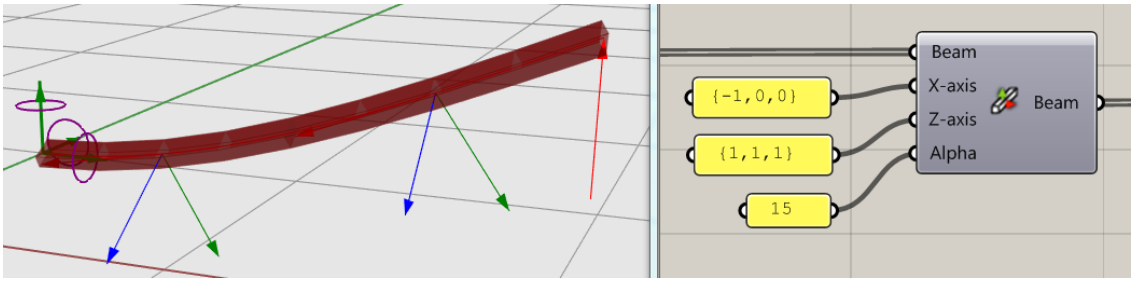


Figure 24: The orientation of the local beam coordinate system can be controlled with the OrientateBeam-component.

In Karamba the default orientation of the local coordinate system of a beam or truss follows these conventions:

- The local X-axis (of red color) is the beam axis and points from starting-node to end-node.
- The local Y-axis (green) is at right angle to the local X-axis and parallel to the global XY-plane. This specifies the local Y-axis uniquely unless the local X-axis is perpendicular to the XY-plane. If this is the case, then the local Y-axis is chosen parallel to the global Y-axis.
- The local Z-axis (blue) follows from the local X- and Y-axis so that the three of them form a right-handed coordinate system.

The local coordinate system affects the direction of locally defined loads and the orientation of the element's cross section. Use the "Orientate Beam" component to set the local coordinate system (see fig. 24):

- The input plug "X-axis" accepts a vector. The local X-axis will be oriented in such a way that its angle with the given vector is less than $90[deg]$. This allows to give a consistent orientation to a group of beams.
- The local Z-axis lies in the plane which is defined by the local X-axis and the vector plugged into the "Z-axis"-input.
- "Alpha" represents an additional rotation angle (in degree) of the local Z-axis about the local X-axis.

6.1.13. Select Beam

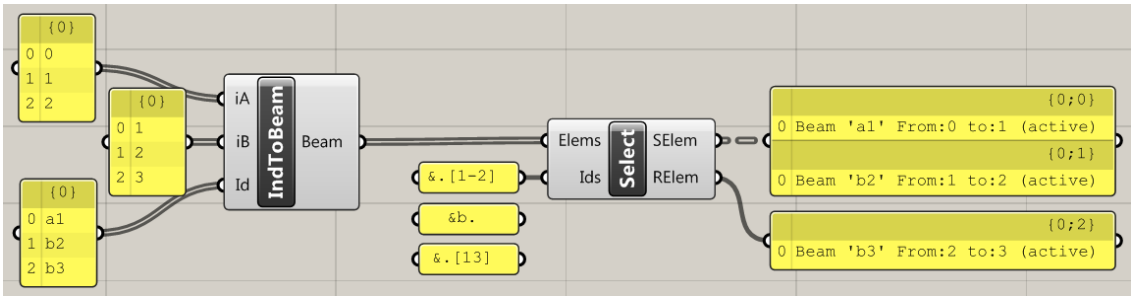


Figure 25: Elements can be selected by using their identifiers.

All structural elements can be given identifiers, i.e. names. These names need not be unique: Two elements can have the same name without Karamba complaining. By default an element identifier corresponds to the elements index. Figure 25 shows how a list of elements can be split into two data trees using their identifiers. The "Select Beam"-component expects a list of elements in "Elems" as well as a list of identifiers or regular expressions in "Id". Regular expressions need to be prefixed by a "&". They represent a very mighty selection tool. In fig. 25 one can see three use-cases:

- "&.[1-2]": a "." matches any character; "[1-2]" matches one character in the range of "1" to "2". This is equivalent to "[12]".
- "&b.": matches any identifier that starts with "b" followed by an arbitrary character.
- "&.[13]": matches any identifier that starts with an arbitrary character followed either by "1" or "3".

There are two output-plugs on the "Select Beam"-component: "SElem" renders the selected elements which match the selection criteria, "RElem"

returns the rest. The entries of the "SElem" and "RElem" output data remember their spot in the original list of elements. Joining them results in the original order of elements.

6.1.14. Support

Without supports a structure would have the potential to freely move around in space. This is not desirable in case of most buildings. The current version of Karamba does statical calculations. This means that there must always be enough supports so that the structure to be calculated can not move without deforming. Thus rigid body modes are forbidden.

When defining the supports for a structure one has to bear in mind, that in three dimensional space a body has six degrees of freedom (DOFs): three translations and three rotations (see figure 26). The structure must be supported in such a way that none of these is possible without invoking a reaction force at one of the supports. Otherwise Karamba will refuse to calculate the deflected state. Sometimes you get results from moveable structures although you should not: The reason for this lies in the limited accuracy of computer-calculations which leads to round-off errors. Sometimes one is tempted to think that if there act no forces in one direction – consider e.g. a plane truss – then there is no need for corresponding supports. That is wrong: What counts is the possibility of a displacement.

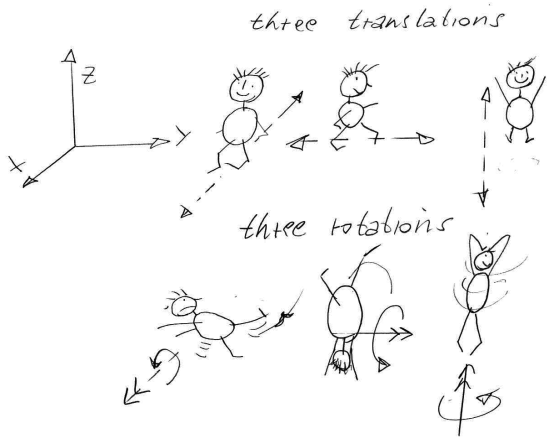


Figure 26: Metaphor for the six degrees of freedom of a body in three-dimensional space.

Bad choices of support conditions are easy to detect with Karamba: In section 6.5.3 it is shown how to calculate the eigen-modes of a structure. This kind of calculation works also in cases of moveable structures: rigid

body modes – if present – correspond to the first few eigen-modes.

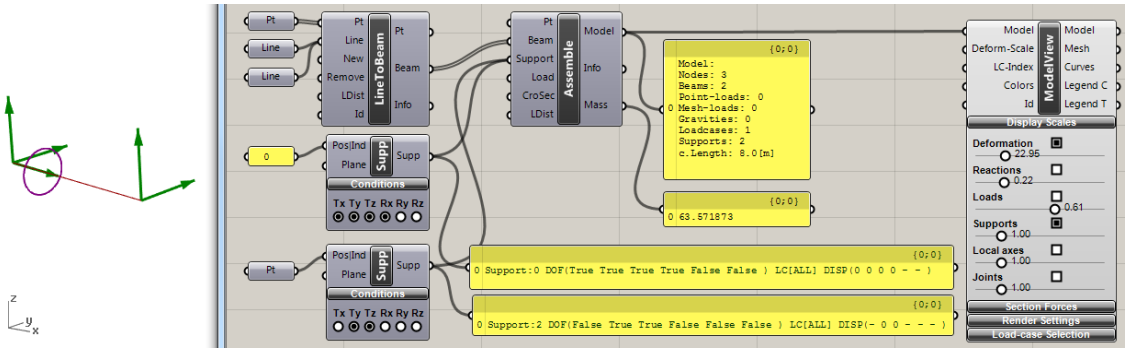


Figure 27: Define the position of supports by node-index or position.

Figure 27 shows a simply supported beam. The "Karamba/Ensemble/-Support"-component takes as input either the index[3] or the coordinates of the point (or a list with indexes or positions of points) to which it applies.

By default the coordinate system for defining support conditions is the global one. This can be changed by defining a plane and feeding it into the "Plane"-input plug of the "Support" component.

Six small circles on the component indicate the type of fixation: The first three correspond to translations in global x, y and z-direction, the last stand for rotations about the global x,y and z-axis. Filled circles indicate fixation which means that the corresponding degree of freedom is zero. The state of each circle can be changed by clicking on it. The string output of the component lists node-index or nodal coordinate, an array of six binaries corresponding to its six degrees of freedom and the number of load-case to which it applies. Supports apply to all load cases by default.

Supports cause reaction forces. These can be visualized by activating "Reactions" in the "Display Scales" section of the ModelView (see section 6.6.2). They show as arrows with numbers in colors green – representing forces – and purple – representing moments. The numbers either mean $[kN]$ in case of forces or $[kNm]$ when depicting moments. The orientation of the moment arrows corresponds to the screw-driver convention: They rotate about the axis of the arrow anti-clockwise when looked at in such a way that the arrow head points towards the observer.

From the support-conditions in figure 27 one can see that the structure is

[3]In order to find out the index of a specific node enable the node-tag checkbox in the ModelView-component. See section 6.1.5 on how to predefine the index of specific nodes

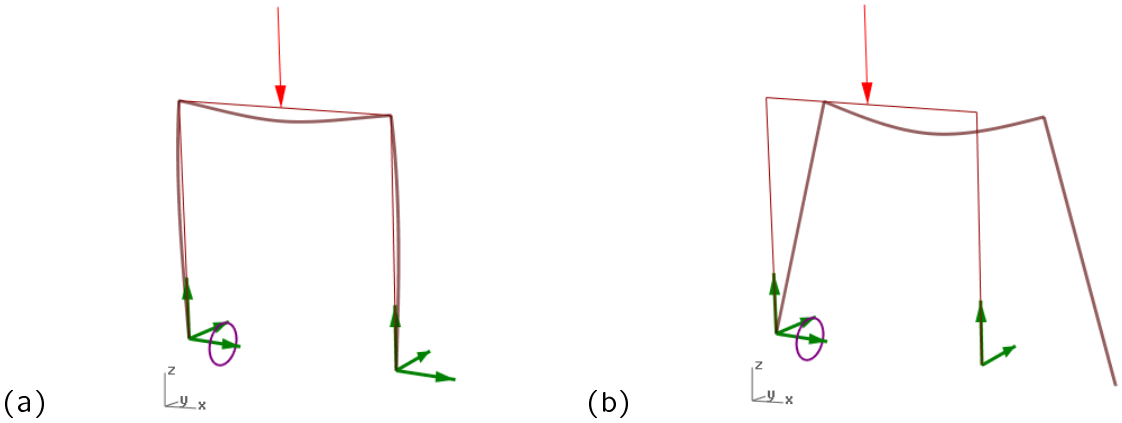


Figure 28: Influence of support conditions – undeflected and deflected geometry. *Left:*All translations fixed at supports. *Right:* One support moveable in horizontal direction.

a simply supported beam: green arrows symbolize locked displacements in the corresponding direction. The translational movements of the left node are completely fixed. At the right side two supports in y- and z-direction suffice to block translational movements of the beam as well as rotations about the global y- and z-axis. The only degree of freedom left is rotation of the beam about its longitudinal axis. Therefore it has to be blocked at one of the nodes. In this case it is the left node where a purple circle indicates the rotational support.

The displacement boundary conditions may influence the structural response significantly. Figure 28 shows an example for this: when calculating e.g. the deflection of a chair, support its legs in such a way that no excessive constraints exist in horizontal direction – otherwise you underestimate its deformation. The more supports you apply the stiffer the structure and the smaller the deflection under given loads. In order to arrive at realistic results introduce supports only when they reliably exist.

By default the size of the support symbols is set to approximately $1.5[m]$. The slider with the heading "Support" on the ModelView-component lets you scale the size of the support symbols. Double click on the knob of the slider in order to set the range of values.

6.2. Cross Section

Karamba offers five basic types of cross section:

- circular tube – the default

- hollow box section
- filled trapezoid section
- I-profile
- shell cross sections

The dimensions of each of these may be defined manually or by reference to a list of cross sections (see section 6.2.9).

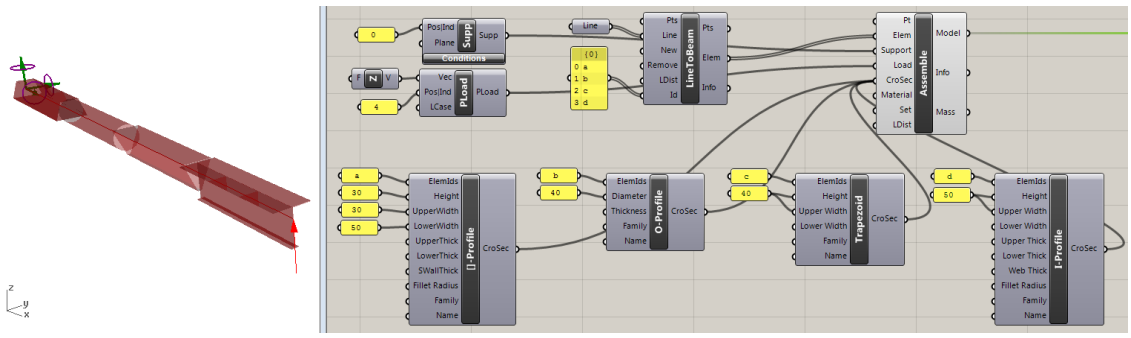


Figure 29: Cantilever with four different kinds of cross section.

Cross sections are autonomous objects which may be plugged into the "Assemble" component (see fig. 29). They know about the elements (or element sets) they belong to by their "ElemIds" property: This is a list of strings containing element identifiers (see 6.1.5) or regular expressions that match a group of element identifiers (elem-ids). Upon assembly all elem-ids are compared to all "ElemIds" entries of a cross section. In case of correspondence the cross section is attached to the element. An empty string – which is the default value – signifies that the cross section shall be applied to all elements. If two cross sections refer to the same element then that which gets processed later by the assemble-component wins. It makes no sense to attribute beam cross sections to shells and vice versa – Karamba ignores any such attempts.

6.2.1. Box-Profile, Circular Profile, I-Profile and Trapezoid-Profile

Fig. 29 shows a cantilever with cross section properties defined by means of beam identifiers. The beam axis always coincides with the center of gravity of a cross section. Changing e.g the upper flange width of an I-section therefore results in a slight movement of the whole section in the local Z-direction. In case the position of e.g. the upper side of a cross section needs to be fixed

specify an eccentricity (see sec. 6.2.6). The corresponding position of the centroid can be retrieved from the "Disassemble Cross Section"-component (see sec. 6.2.5).

Apart from the input-plugs that define the cross section geometry there are the "Family"- and "Name"-plug:

- "Family": Each cross section belongs to a family. When doing cross section optimization (see section 6.5.7), Karamba selects only profiles that belong to the same family as the original section. Families can be composed of arbitrary section types.
- "Name": the identifier of a cross section – need not be unique. Enable "CroSec names" in ModelViews "RenderSettings"-submenu in order to view them.

6.2.2. Shell Cross Section

Each element of a shell can be given an individual, constant thickness. Fig. 30 shows a shell consisting of two elements. Meshes form the basis for defining a shell geometry (see section 6.1.8) and specify the sequence of faces (i.e. shell elements). The list of element thicknesses in fig. 30 corresponds to that order. In case that there are more mesh faces than thickness specifications the last value ($6[cm]$ in this case) acts as the default value. Make sure to graft the "Heights" input when you want to define a list of shell cross sections. Otherwise one cross section results where one would expect several.

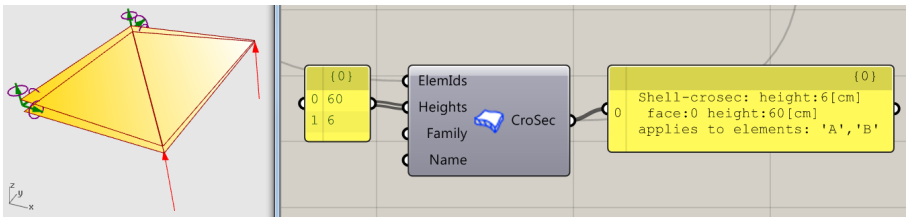


Figure 30: Shell made up of two elements with different thicknesses.

When rendering the shell cross sections (see fig. 30) thicknesses get linearly interpolated between the nodes. The cross section height at each node results from the mean thickness of shell elements attached to it.

The input-plugs "Family" and "Name" have the same meaning as described in section 6.2.1.

6.2.3. Spring Cross Section

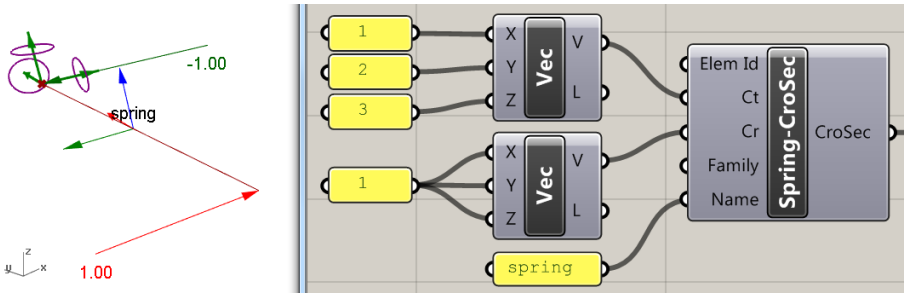


Figure 31: Spring fixed at one end and loaded by a point load on the other.

Springs allow you to directly define the stiffness relation between two nodes via spring constants. Each node has six degrees of freedom (DOFs): three translations and three rotations. Using the "Spring-CroSec"-component lets one couple these DOFs by means of six spring-constants. A relative movement $u_{i,rel}$ between two nodes thus leads to a spring force $F_i = c_i \cdot u_{i,rel}$. In this equation $u_{i,rel}$ stands for a relative translation or rotation in any of the three possible directions x, y, z, c_i is the spring stiffness. In Karamba the latter has the meaning of kilo Newton per meter $[kN/m]$ in case of translations and kilo Newton meter per radiant $[kNm/rad]$ in case of rotations. The input-plugs "Ct" and "Cr" expect to receive vectors with translational and rotational stiffness constants respectively. Their orientation corresponds to the local beam coordinate system to which they apply. In case of zero-length springs this defaults to the global coordinate system but can be changed with the "OrientateBeam"-component.

In case one wants to realize a rigid connection between two nodes the question arises as to which spring stiffness should be selected. A value too high makes the global stiffness matrix badly conditioned an can lead to a numerically singular stiffness matrix. A value too low results in unwanted relative displacements. So you have to find out by trial and error which value gives acceptable results.

In Karamba the definition of a spring is analogous to creating a cross section. This is why the "Spring-CroSec"-component offers the possibility to give springs a cross section family and name. The "SpringCroSec"-component attaches spring properties to beams via their identifiers.

Figure 31 shows a peculiarity one has to be aware of when using springs: They are unaware of the relative position of their endpoints. This is why the load on the right end of the spring does not evoke a moment at the left,

fixed end of the spring.

6.2.4. Beam Joints

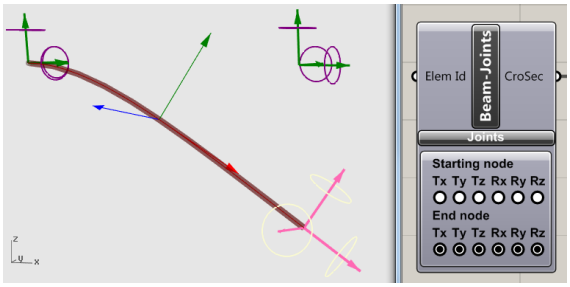


Figure 32: Beam under dead weight, fixed at both supports with a fully disconnected joint at one end resulting in a cantilever.

A structure usually consists of a large number of load bearing elements that need to be joined together. When rigidly connected such a joint has to transfer three section forces (one axial force, two shear forces) and three moments (one torsional and two bending moments). Depending on the type of material such full connections are sometimes (e.g. for wood) hard to achieve, costly and bulky. A solution to this problem consists in introducing hinges.

Figure 32 shows a beam under dead weight with fully fixed boundary conditions at both end-points. At the right end the joint (which is in fact no joint any more) completely dissociates the beam from the support there. The result is a cantilever.

The symbols for joints resemble that for supports: pink arrows represent translational joints, white circles symbolize moment hinges. In Karamba joints are realized by inserting a spring between the endpoint of a beam and the node to which it connects. This necessitates sufficient support conditions at the actual nodes to prevent them from freely moving around. See for example the right node in fig. 32 which has to be fully fixed – otherwise the system would be kinematic.

The "Crosec-Joint"-component allows to define hinges at a beams starting- and end-node. A list of beam-identifiers lets you select the beams where the joint definition shall apply: Filled circles mean that the corresponding degrees of freedom represent joints. "T" stands for translation, "R" for rotation. Feed the resulting cross-section into the "CroSec"-plug of the "Assemble"-component.

The orientation of the axes of the joints corresponds to the local coordinate system of the beam they apply to.

6.2.5. Disassemble Cross Section

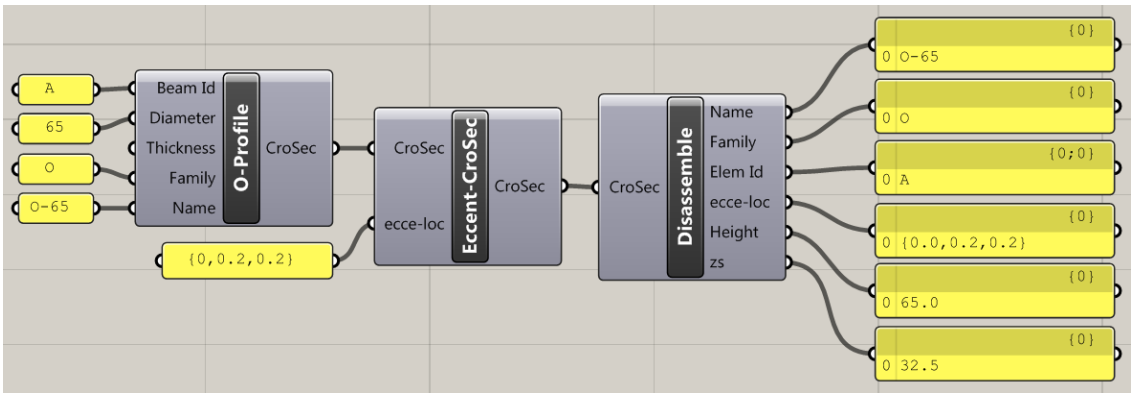


Figure 33: Properties of a given cross section can be retrieved via the "Disassemble Cross Section"-component.

In some cases (e.g. after optimizing cross sections) it may be necessary to retrieve the properties of a cross section. Use the "Disassemble Cross Section"-component for that (see fig. 33).

6.2.6. Eccentricitiy on Beam, Eccentricity on Cross Section

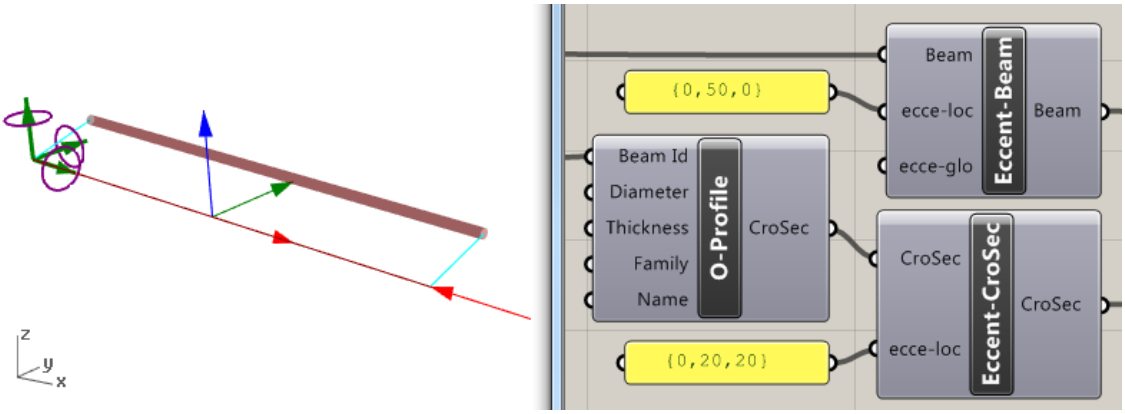


Figure 34: Beam positioned eccentrically with respect to the connection line of its two end-nodes.

Cross section forces of beam and truss elements relate to the line that connects the cross section centroids. When a cross section changes, chances are

high that also the position of its centroid shifts. In case of elements predominantly loaded by bending moments, such a shift can normally be neglected. In the presence of normal forces however – e.g. when considering columns – changes in the centroids position lead to additional bending moments that may be decisive for a members cross section design.

In Karamba there exist two components that can be used to take care of eccentricities (see fig. 34): One works on beams, the other on cross sections. When both variants of definition coincide for an element then they get additively combined. This enables one to define families of cross sections of different size with e.g. the position of their upper sides at one level.

The definition of a local eccentricity for cross sections with a "Eccent-CroSec"-component is straight forward: The "ecce-loc"-input plug expects a vector that defines the offset with respect to the local beam axes. Values are expected in centimeters. "x" represents the longitudinal beam axis, "y" is horizontal, "z" vertically upwards. Cross sections with eccentricities can be stored in cross section tables using the "GenCSTable"-component and thus be made reusable in other projects.

The "Eccent-Beam"-component has one additional input-plug as compared to the cross section variant: "ecce-glo" lets one define beam eccentricities ($[cm]$) with respect to the global coordinate system.

6.2.7. Cross Section Matcher

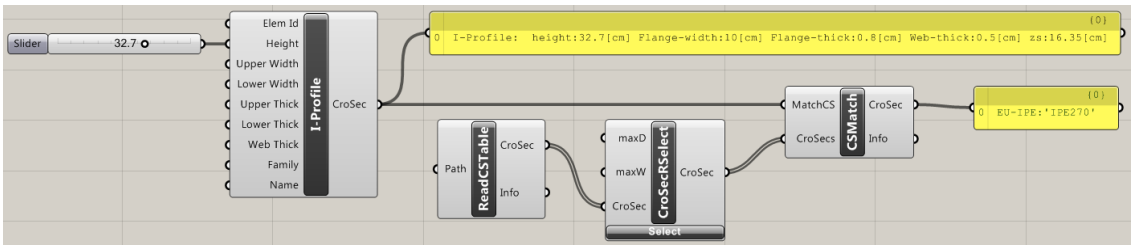


Figure 35: The "Cross Section Matcher"-component returns a standard profile for a custom profile.

Use the "Cross Section Matcher"-component in case you want to find the first profile from a given list that provides equal or higher resistance compared to a given custom profile (see fig. 35). The "CSMatch"-component takes a cross section and a list of cross sections as input. Traversing the list starting from the first element it proceeds until an appropriate profile is found which is returned as the result.

6.2.8. Cross Section Range Selector

The cross section library that comes with Karamba contains roughly 5300 profiles. In order to reduce the amount of information the list can be shortened by applying selection criteria on it using the "Cross Section Range Select" component (see fig. 36). The input-plugs "maxH" and "maxW" let you limit the list according to maximum cross section height and width. The submenu which unfolds when clicking on the black "select"-bar offers further options for narrowing the search: country of origin, general shape and family name.

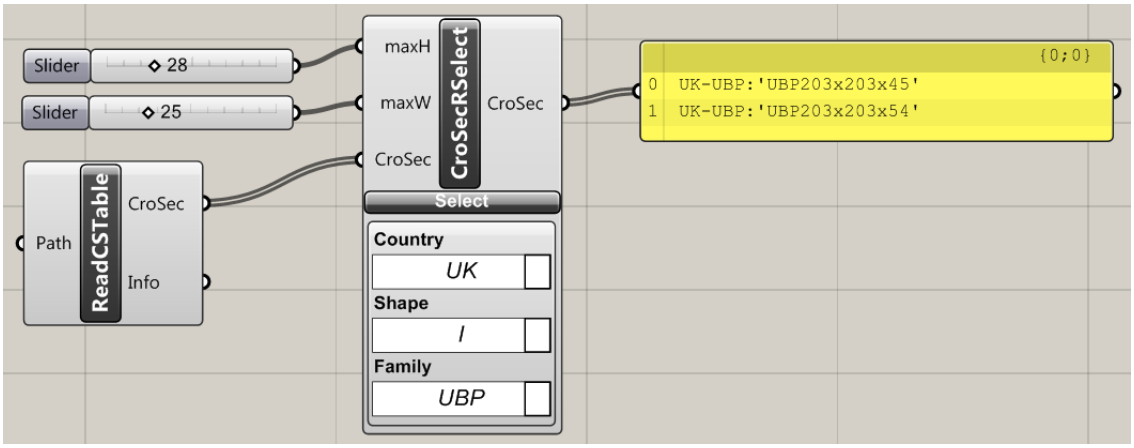


Figure 36: Selection of a range of cross sections from among a given list.

6.2.9. Cross Section Selector

The component "CroSecSelect" deals with selecting cross sections by name or index from a list of cross sections. Provide the name(s) or index(es) of desired cross sections in the "Name|Ind" plug. Cross section names are not case sensitive. All characters coming after "#" count as remark. It is possible to use regular expressions for selection. In this selection is case sensitive and "&" has to be provided as first character. List indexes start from zero.

"CroSecSelect" lets you specify beams via the "Elem Id"-plug which shall be assigned a specific cross section. The "Assemble"-component sets the cross-sections of elements accordingly.

6.2.10. Generate Cross Section Table

An entry in a cross section table consists of a row which contains:

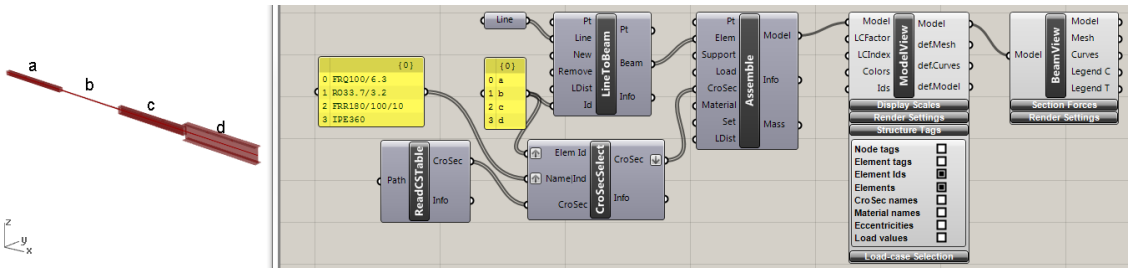


Figure 37: Cantilever with four different kinds of cross section taken from the standard cross section table.

- "country": country of origin
- "family": name of the group to which the cross section belongs (see sec. 6.2.1)
- "name": name of the specific cross section (see sec. 6.2.1)
- a "shape" field which defines the basic cross section type:
  - "I": I-section
  - "[]": hollow box section
  - "V": trapezoid, filled section
  - "O": circular tube
  - "S": spring
  - "Sh": shell
- geometric properties which are used for drawing the cross section
- area, moments of inertia, etc. that define the cross sections mechanical behavior.

The "GenCSTable"-component takes a cross section (or a list of cross sections) as input and returns the equivalent table data as a string. The physical units used for output are always metric. When plugged into a panel the information can be streamed to a file which then constitutes a valid cross section table. Karamba reads the data of cross section tables only once. So in order that changes in the table take effect restart Grasshopper.

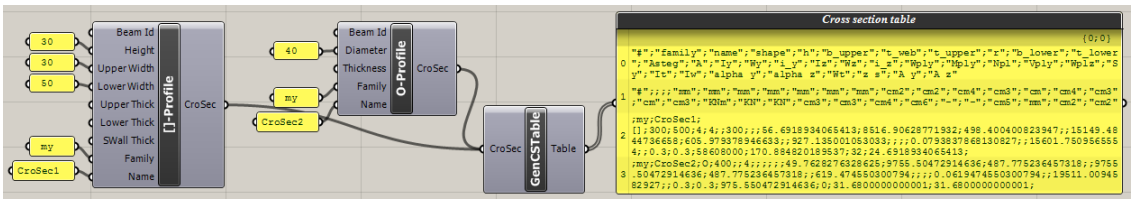


Figure 38: Transformation of a list of cross sections to a cross section table.

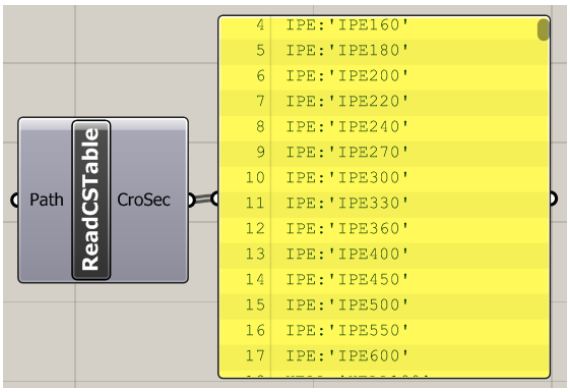


Figure 39: List of cross sections generated from the standard cross section table.

6.2.11. Read Cross Section Table from File

Predefined cross sections stored in a csv-database can be used to generate lists of cross sections via the "ReadCSTable"-component (see fig. 39). It works along the same lines as the "ReadMatTable" (see section 6.3.3) component. When given no path to a valid csv-table "ReadCSTable" uses the cross section table that comes with Karamba and is situated in ".../-Grasshopper/Libraries/Karamba/CrossSectionValues.csv". This table contains definitions for a range of standard steel profiles. Use a text editor or OpenOffice to view or extend the table. "#" is used to mark the rest of a line as comment. The physical units are always assumed to be metric – irrespective of the user settings at installation.

When opening the "Karamba"-folder you will find three differently named cross section tables: "CrossSectionValues.csv" and "CrossSectionValues_sortedForHeight.csv" contain cross sections sorted according to increasing height. In "CrossSectionValues_sortedForWeight.csv" the area and thus weight per unit of length determines a cross sections relative position within a family. When doing cross section optimization (see section 6.5.7) those two sorting options lead to different results. Depending on external requirements they

result in structures of minimum cross section height or structural weight.

6.3. Material

There are two ways for defining materials in Karamba: Either select a material by name from a list of materials (see section 6.3.2) or set mechanical material properties manually (see below).

The Appendix (see section A.1) contains additional information on mechanical properties of materials.

Materials (like cross sections) are autonomous entities which may be plugged into the "Assemble" component. They know about the elements (or element sets) they apply to by their "ElemIds" property: This is a list of strings containing element identifiers (see 6.1.5) or regular expressions that match a group of element identifiers (element-ids). Upon assembly each element-id is compared to all "ElemIds" entries of a material. In case they match the material is attached to the element. An empty string – which is the default value – signifies that the material shall be applied to all elements.

6.3.1. Material Properties

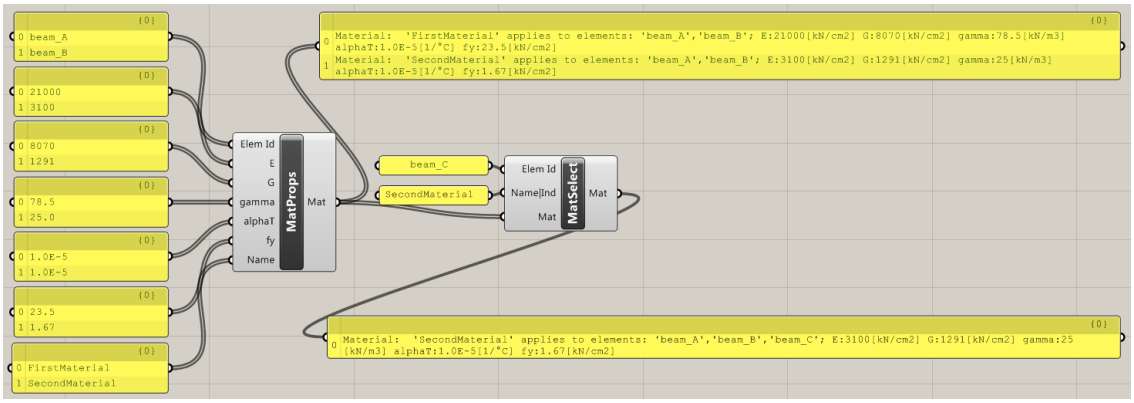


Figure 40: The definition of the properties of two materials via the MatProps component and selection of the second Material from the resulting list.

The component "MatProps" lets one directly define material properties (see fig. 40):

- "ElemIds": list of identifiers of elements or a regular expression that depicts the elements that shall have the specified material
- Young's Modulus "E" $[kN/cm^2]$

- Shear modulus "G" $[kN/cm^2]$
- specific weight "gamma" $[kN/m^3]$
- coefficient of thermal expansion "alphaT" $[1/°C]$
- yield stress "fy" $[kN/cm^2]$
- the name of the material in "Name"

The yield stress characterizes the strength of a material. The utilization of cross sections as displayed by the BeamView (see section 6.6.8) is the ratio of actual stress and yield stress. In case of shells, utilization is determined as the ratio of Van Mises Stress and yield stress (see section 6.6.18). Cross section optimization (see section 6.5.7) also makes use of the materials yield stress.

In case of temperature changes materials expand or shorten. "alphaT" sets the increase of strain per degree Celsius of an unrestrained element. For steel the value is $1.0E-5$ ($1.0E-5 = 1.0 \cdot 10^{-5} = 0.00001$). Therefore an unrestrained steel rod of length $10[m]$ lengthens by $1[mm]$ under an increase of temperature of $10°C$. "alphaT" enters calculations when temperature loads are present.

In order to be registered with the model the resulting material needs to be plugged into the "Assemble Model"-component.

6.3.2. Material Selection

The "MatSelect"-component in the menu subsection "Material" lets you select a material by name or index from a given list of materials (see fig. 40 or fig. 42). The names of materials are not case sensitive. A "#" in a material name means that the rest of the line is a comment. "&" starts a regular expression – in that case material names are case sensitive. "Mat" expects a list of materials, the input-plug "Name" material names. Element identifiers given through the "MatSelect"-component add to those already given for a material (see fig 40).

6.3.3. Read Material Table from File

Karamba comes with a table of predefined materials. The csv-file "Material-properties.csv" resides in the "Karamba"-folder inside the "Libraries" directory . By default the "ReadMatTable"- component takes this file and creates a

	A	B	C	D	E	F	G	H
1	#			E	G	gamma	alphaT	fyd
2	#	family	name	[kN/cm2]	[kN/cm2]	[kN/m3]	[m/(mC°)]	[kN/cm2]
3		Steel	Steel	21000	8076	78.5	1.20E-05	23.5
4		Steel	S235	21000	8076	78.5	1.20E-05	23.5
5		Steel	S275	21000	8076	78.5	1.20E-05	27.5
6		Steel	S355	21000	8076	78.5	1.20E-05	36
7		Steel	St37-2	21000	8076	78.5	1.20E-05	24
8	#							

Figure 41: Partial view of the default data base of materials. SI units are used irrespective of user settings. Automatic conversion ensures compatibility with Imperial units.

list of materials from it. These are available at the output-plug "Material". The data-base currently holds properties for "steel", "concrete", "wood" and "aluminum". There exist different types of steel, concrete etc.. The generic term "concrete" will result in the selection of an everyday type of concrete - a C25/30 according to Eurocode. More specific descriptions may be given: Have a look at the data-base in order to get an overview. Material properties specified via table are assumed to be in SI units. They get automatically converted when used in the context of Imperial units.

The extension .csv stands for "comma separated value". The file can be opened with any text editor and contains the table entries separated by semicolons. It is preferable however to use OpenOffice or Excel (both can read and write csv-files): They render the data neatly formatted (see fig. 41). Make sure to have a "." and not a "," set as your decimal separator. In some countries "." is used to separate thousands which then needs to be adapted as well. The setting may be changed under Windows via "regional settings" in "system settings". All lines in the table that start with "#" are comments. Feel free to define your own materials. All physical units in the table are assumed to be metric irrespective of the user choice at installation.

The file path to the materials data-base can be changed in two ways: first right-click on the component and hit "Select file path to material definitions" in the context menu that pops up. Second plug a panel with a file path into "Path". Relative paths are relative to the directory where your definition lies.

6.4. Load

Currently Karamba supports seven kinds of loads: point-, mesh-, gravity-, uniformly distributed-, pretension-, temperature-loads and prescribed displacements at supports. An arbitrary number of point-, mesh-, etc.-loads and one gravity-load may be combined to form a load-case of which again an arbitrary number may exist. Figure 43 shows the definition of loads with

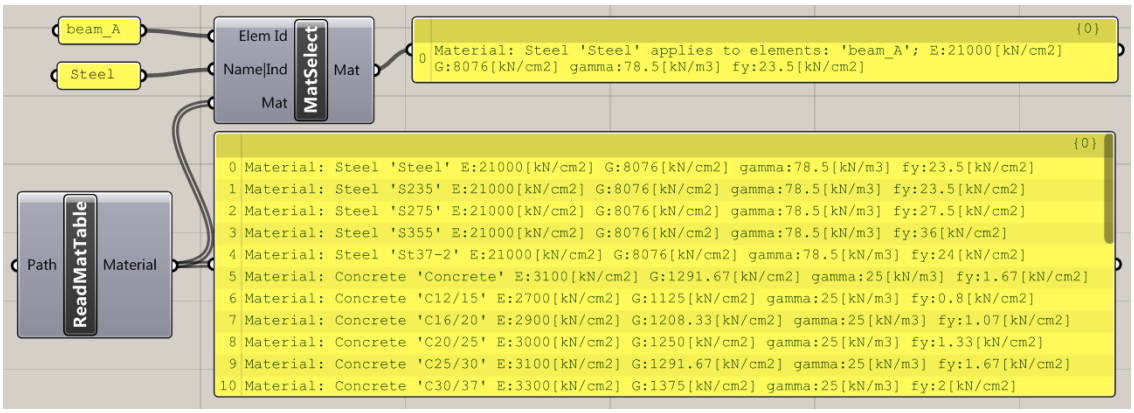

beam_A
Steel
Elem Id
Name|Ind
Mat
MatSelect
Mat
{0}
0 Material: Steel 'Steel' applies to elements: 'beam_A'; E:21000[kN/cm2] G:8076[kN/cm2] gamma:78.5[kN/m3] fy:23.5[kN/cm2]
Path
ReadMatTable
Material
{0}
0 Material: Steel 'Steel' E:21000[kN/cm2] G:8076[kN/cm2] gamma:78.5[kN/m3] fy:23.5[kN/cm2]
1 Material: Steel 'S235' E:21000[kN/cm2] G:8076[kN/cm2] gamma:78.5[kN/m3] fy:23.5[kN/cm2]
2 Material: Steel 'S275' E:21000[kN/cm2] G:8076[kN/cm2] gamma:78.5[kN/m3] fy:27.5[kN/cm2]
3 Material: Steel 'S355' E:21000[kN/cm2] G:8076[kN/cm2] gamma:78.5[kN/m3] fy:36[kN/cm2]
4 Material: Steel 'St37-2' E:21000[kN/cm2] G:8076[kN/cm2] gamma:78.5[kN/m3] fy:24[kN/cm2]
5 Material: Concrete 'Concrete' E:3100[kN/cm2] G:1291.67[kN/cm2] gamma:25[kN/m3] fy:1.67[kN/cm2]
6 Material: Concrete 'C12/15' E:2700[kN/cm2] G:1125[kN/cm2] gamma:25[kN/m3] fy:0.8[kN/cm2]
7 Material: Concrete 'C16/20' E:2900[kN/cm2] G:1208.33[kN/cm2] gamma:25[kN/m3] fy:1.07[kN/cm2]
8 Material: Concrete 'C20/25' E:3000[kN/cm2] G:1250[kN/cm2] gamma:25[kN/m3] fy:1.33[kN/cm2]
9 Material: Concrete 'C25/30' E:3100[kN/cm2] G:1291.67[kN/cm2] gamma:25[kN/m3] fy:1.67[kN/cm2]
10 Material: Concrete 'C30/37' E:3300[kN/cm2] G:1375[kN/cm2] gamma:25[kN/m3] fy:2[kN/cm2]

Figure 42: List of materials resulting from the "ReadMatTable"-component reading the default data base of materials. Selection of the default "Steel" via "MatSelect".

the help of Gravity- and Point-Load components. On the bottom of the ModelView-component (see section 6.6.2) there is a drop-down-list (unfold it by clicking on the "Load-case Selection"-menu header) which can be used to select single load-cases for display. Select "–all–" in order to view all existing load-definitions of all load-cases simultaneously. Use the force-slider to scale the size of the load-symbols (double-clicking on its knob lets you change the value range and its current value).

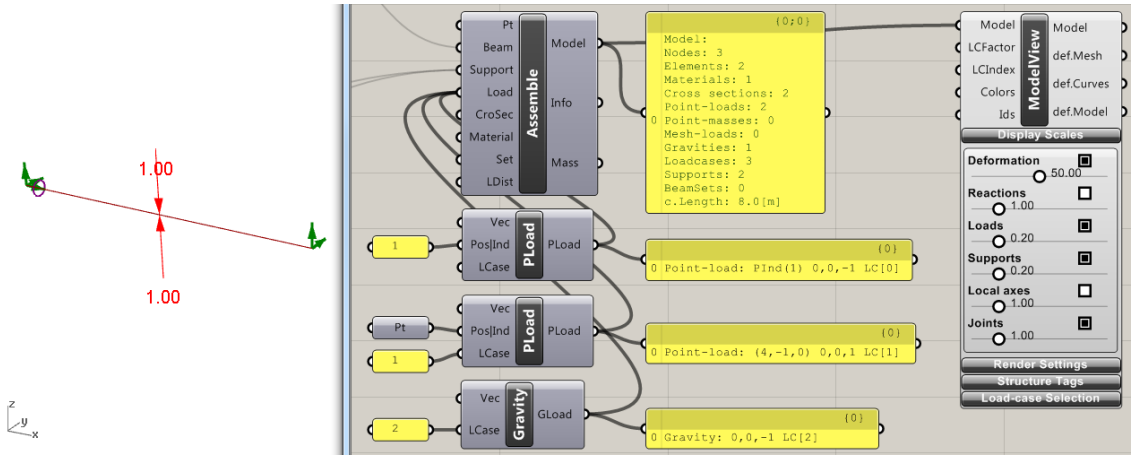


Figure 43: Simply supported beam with three loads and three load-cases.

6.4.1. Point-Load

The component "Point-Load" lets you define loads on points. These get attached to their points either by node-index[6] or coordinate. Feed a corresponding list of items into the "Pos|Ind"-plug (quite analogous to the Support-component). A point-load is given as a vector. Its components define the force-components in global x-, y- and z-direction. The output of the "force"-component has to be connected to the corresponding plug of the assembly. Plugging a point-load into a panel component gives the following information: Node-index where the load gets applied, force-vector and number of the load case to which it belongs.

By default point loads will be put into load case zero. Any positive number fed into the "LCase"-plug defines the load case to which the corresponding load will be attributed. A value of -1 signals that the load acts in all existing load cases.

For more information on loads and some typical values see section A.2.

6.4.2. Mesh-Load

The Mesh-load-component can be used to transform surface loads into equivalent node- or element-loads. This lets you define life-loads on floor slabs, moving loads on bridges (see example "Bridge.ghx" in the examples collection on the Karamba web-site), snow on roofs, wind-pressure on a facade, etc.. Figure 44 left side shows a simply supported beam and a mesh which consists of two rectangular faces. Each face covers one half of the beam and has a width of $2[m]$ perpendicular to the beam axis. With a distributed load of $1[kN]$ in negative global Z-direction a uniformly distributed load of $2[kN/m]$ results.

With the input-plug "BeamIds", groups of elements can be specified on which equivalent loads shall be generated. By default all beams of the model are included. In case no beam loads shall be included provide an identifier like e.g. "None" which does not correspond to an element in the model.

In order to define structure nodes where equivalent point-loads may be generated, plug a list of their coordinates into the "Pos"-plug. These need to correspond to existing nodes – otherwise the Assembly-component turns

[6] In order to find out the index of a specific node enable the "node tag"-checkbox in the ModelView-component. See section 6.1.5 on how to predefine the index of specific nodes or node-position

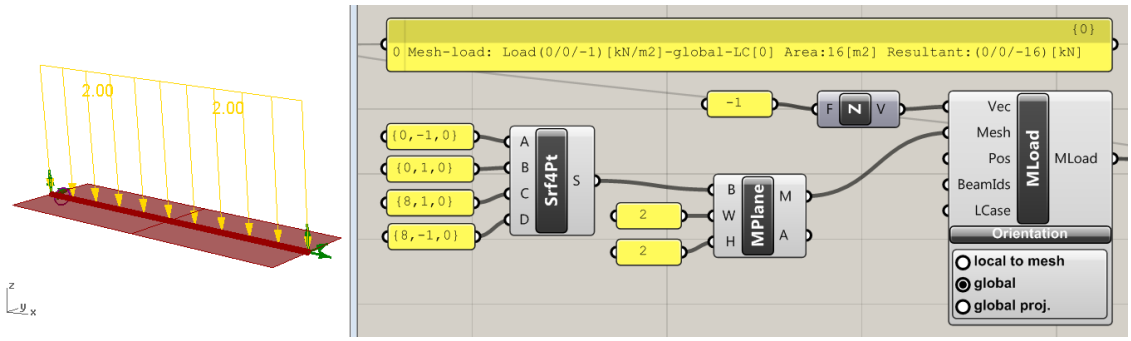


Figure 44: Simply supported beam loaded with line loads that approximate a given, evenly distributed surface load on a mesh.

red. Offending nodes will be listed in its run-time error message. By default no point-loads will be produced.

The procedure for calculating nodal loads and uniformly distributed beam loads from surface loads consists of the following steps: First Karamba calculates the resultant load on each face of the given mesh. Then the resultant load of each face gets evenly distributed among its vertices.

The second step consists of distributing the vertex-loads among the nodes of the structure. In order to arrive at beam loads additional helper-nodes along their axes get generated. The mutual distance of those is chosen equal to a third of the mean edge length of the given mesh.

Each mesh vertex transfers its load to the nearest node. In case that there are several nodes within a radius of less than "LDist" as set at the Assemble-component (see section 6.1.2) the vertex load gets evenly distributed among them. The loads received by the helper-nodes along beam axes get summed up and divided by the element length. This results in the approximately equivalent uniformly distributed load which is placed on the element. From the procedure described, one can see that a crude mesh may lead to a locally incorrect distribution of loads.

Fig. 45 shows a similar setting as in fig. 44. The difference lies in the additional input of node positions for the generation of point loads. These points lie at the end- and center-points of the beam. Their distance to the mesh-vertices is smaller than that of the collocation points on the beam-axis. Thus all the mesh-load gets reduced to point-loads. A finer resolution of the loaded mesh would remedy that.

The right side of figure 45 shows what data the Mesh-load-component collects: The input-plug "Vec" expects a vector or list of vectors that define

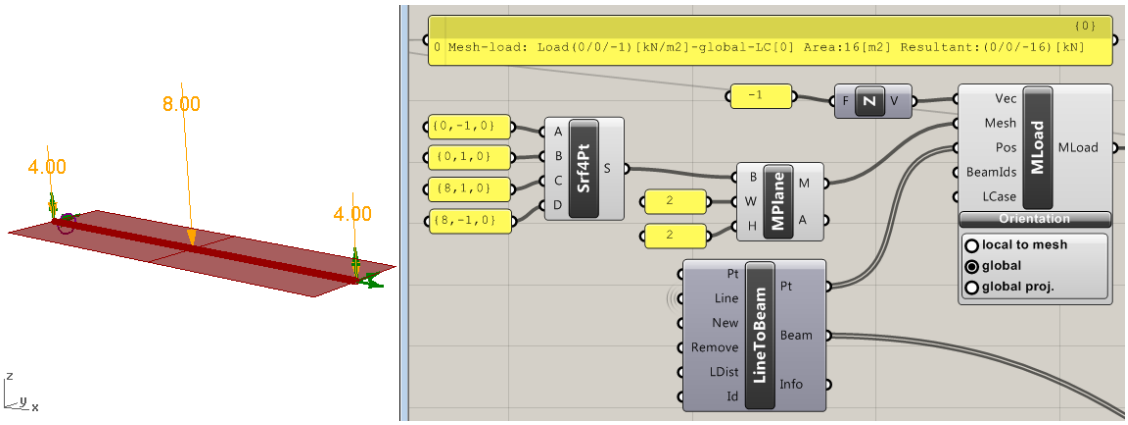


Figure 45: Simply supported beam loaded with point loads (dark orange) that approximate a given, evenly distributed surface load on a mesh.

the surface load. Its physical units are kilo Newton per square meter (kN/m^2). The orientation of the load-vector depends on the checkbox selected under "Orientation" (see also figure 46):

- "local to mesh": X-component of the force vector is at right angle to the mesh-face; the Y-component acts horizontally if the mesh-face X-axis is not parallel to the global Z-axis. Otherwise the Y-component of the force is parallel to the global Y-axis. This means a surface load with components only in X-direction acts like wind pressure.
- "global": The force-vector is oriented according to the global coordinate system. This makes the surface load behave like additional weight on the mesh plane.
- "global proj.": The force-vector is oriented according to the global coordinate system. The corresponding surface load is distributed on the area that results from projecting the mesh-faces to global coordinate planes. In such a way the action of snow load can be simulated.

The input-plug "Mesh" accepts the mesh where the surface load shall be applied. Its vertices need not correspond to structure nodes. The mesh may have any shape.

Set the "LCase"-input to the index of the load case in which the surface load shall act. Indexing of load-cases starts with zero, "-1" is short for all load cases.

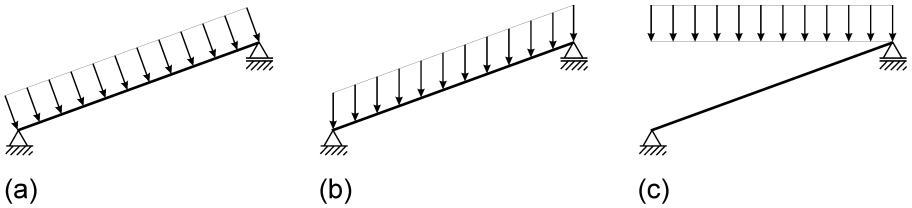


Figure 46: Orientation of loads on mesh: (a) local; (b) global; (c) global projected to global plane.

6.4.3. Line-Load on Element

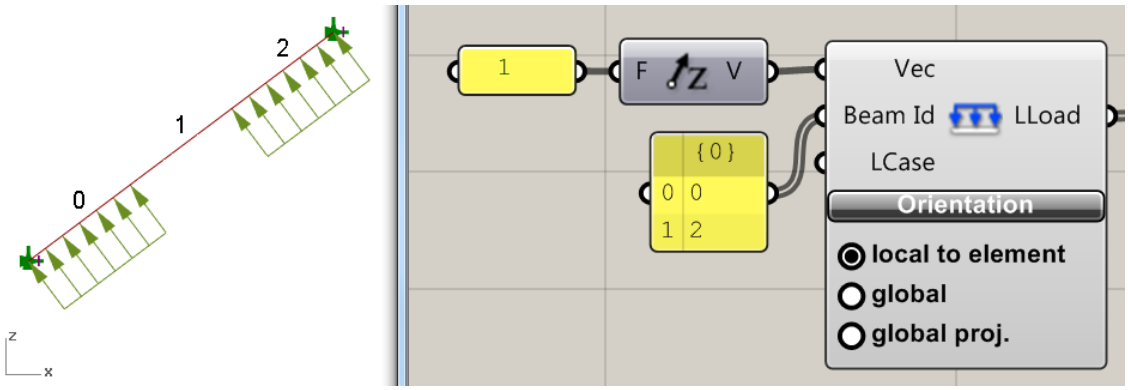


Figure 47: Line loads on a structure consisting of three beam elements defined in local beam coordinate systems.

Figure 47 shows a tilted structure consisting of three beams under the action of a uniformly distributed load at elements "0" and "2". The load acts parallel to the beams local z-axis. The components of the load vector are assumed to be given in kilo Newton per meter $[kN/m]$. The input-plug "BeamIds" receives a list of the identifier of the beams on which the load shall act. See section 6.1.5 for how to attach identifiers to beams. By default beams are named after their index in the FE-model. There are three options for the orientation of the load: "local to element", "global" and "global proj.". Their meaning corresponds to the options available for mesh-loads (see fig. 46). The input-plug "LCase" which designates the load case defaults to "0".

6.4.4. Pretension-Load

Karamba lets you define axial pretension. Pre-tensioning means that the element gets first axially loaded in such a way that it reaches the target strain. Then it gets built into the structure. Fig. 48 shows a beam with

both ends fixed, subject to a compressive pretension load. The unit of dimension of the pretension which gets fed into the "eps0" plug is $[mm/m]$.

Pre-tensioning an element is not the same as applying a pair of opposite forces at its endpoints: In case of pretension the axial force in the element depends on its boundary conditions: If the structure to which it connects is very stiff then the resulting axial force will be $N = \epsilon_0 \cdot A \cdot E$. In figure 48 the supports are rigid, the elements cross section $A = 25[cm^2]$, Young's Modulus $E = 21000[kN/cm^2]$ and $\epsilon_0 = -0.00015$. This results in an axial force of $N = -78.75[kN]$ and shows up as horizontal support reactions. When the rest of the structure does not resist, then a pretension-load merely results in lengthening or shortening the corresponding element.

The input plug "ElemIds" defines the elements where the load acts and "LCase" the load-case.

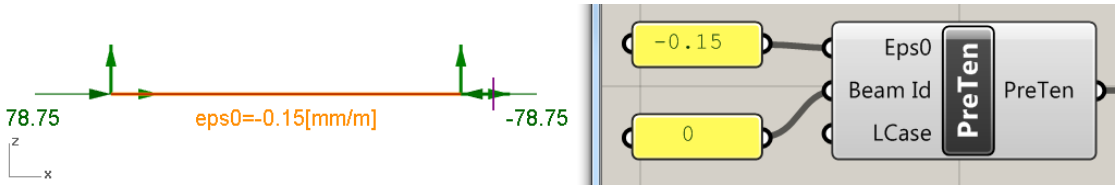


Figure 48: Pre-tensioned member fixed at both ends and resulting support reactions.

6.4.5. Temperature-Load

The definition of temperature loads works analogously to defining pretension loads (see sec. 6.4.4). The difference is, that members get built into the structure first, then they are subjected to a temperature change. The coefficient of thermal expansion (see section 6.3.1) characterizes the response of a materials to temperature changes.

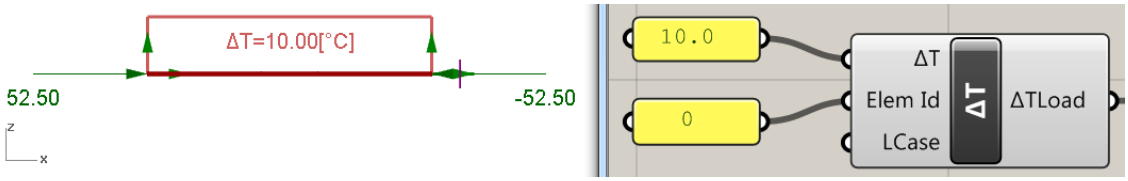


Figure 49: Temperature load on a member which is fixed at both ends.

6.4.6. Gravity

Each load case may contain zero or one definition for the vector of gravity. In this way one can e.g. simulate the effect of an earthquake by applying a certain amount of gravity in horizontal direction. For Vienna which has medium earthquake loads this amounts to approximately 14% of gravity that a building has to sustain in horizontal direction. In areas with severe earthquake loads this can rise to 100% (this however also depends on the stiffness properties of the structure and underlying soil).

The gravity component applies to all active elements in the statical model for which gamma (see section 6.3.1) is not zero. The gravity vector defines the direction in which gravity shall act. A vector of length one corresponds to gravity as encountered on earth.

6.4.7. Point-Mass

Karamba is capable of calculating the natural vibration modes and frequencies of structures (see sec. 6.5.4). For results to match reality the inertia properties of a structure need to be modeled correctly. Masses of elements (e.g. beams, trusses, shells) are automatically taken care of. All other items need to be included via point-masses. Be aware of the fact that masses defined with the "Point-Mass"-component do not have a weight but inertia only! Thus they only show effects in connection with the calculation of natural frequencies. The "Point-Mass" component expects a mass in $[kg]$ at its input-plug "Mass" (see fig. 50). Nodes where masses shall sit can be identified by supplying node indexes or positions (just like for point-loads). Point masses get displayed as green spheres. Their diameters result from the volume calculated as mass divided by density. The latter defaults to $7850[kg/m^3]$ (steel) and can be provided at the input-plug-"rho".

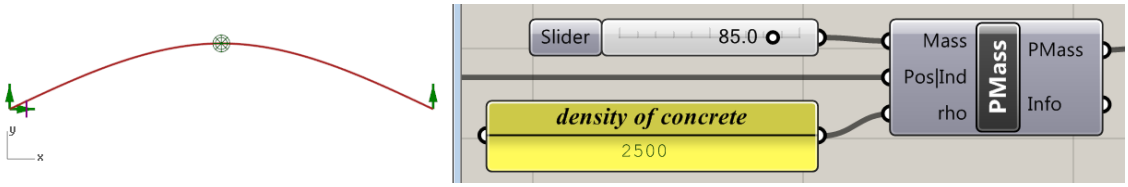


Figure 50: Vibration mode of beam with point mass in the middle.

6.4.8. Prescribed displacements

Supports as described in section 6.1.14 are a special case of displacement[4] boundary conditions: They set the corresponding degree of freedom of a node to zero. The more general PreDisp-component lets you preset arbitrary displacements at nodes. Figure 51 shows a beam with prescribed, clockwise rotations at both end-points.

The PreDisp-component resembles the Support-component to a large degree: Nodes where displacement conditions apply can be selected via node-index[5] or nodal coordinates. The "Plane"-plug can be used to define an arbitrarily oriented coordinate system for the application of support conditions.

Input-plug "LCase" lets you set the index of the load-case in which displacements shall have a specified value. The default value is "-1" which means that the displacement condition is in place for all load-cases. It is not possible to have displacement boundary conditions active in one load-case and completely disabled in others: For load-cases not mentioned in "LCase" the PreDisp-component will act like a simple support with fixed degrees of freedom equal to zero.

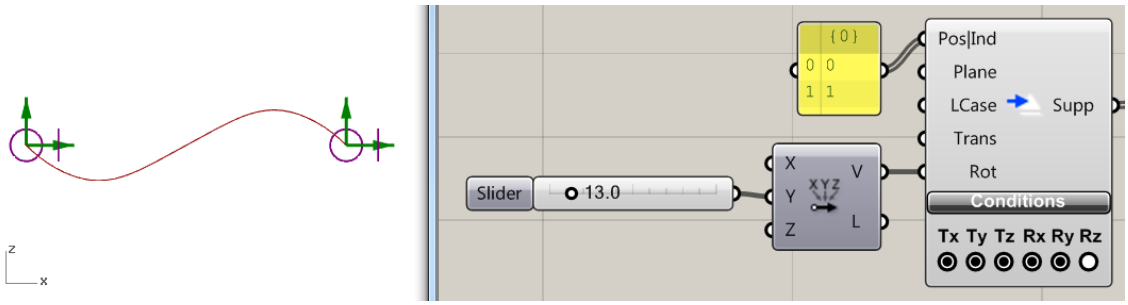


Figure 51: Left: Deflection of a beam under predefined displacements at its end-supports; Right: PreDisp-component for setting displacement condition at left support.

The "Trans"- and "Rot"-input-plugs expect vectors. They define nodal translations and rotations either in global coordinates or in the coordinate system defined by the plane fed into the "Plane"-input plug. Translations are to be given in meter, rotations in degree. The X-component of the rotation vector describes a rotation about the coordinate systems X-axis.

[4] The term "displacement" as used throughout this manual includes translations and rotations.

[5] In order to find out the index of a specific node enable the "node tag"-checkbox in the ModelView-component. See section 6.1.5 on how to predefine the index of specific nodes

A positive value means that the node rotates counter-clockwise if the X-axis points towards you. Analog definitions apply to rotations about the Y- and Z-axis. Karamba is based on the assumption of small deflections. Thus be aware that large prescribed displacements and rotations give rise to incorrect results (which can nevertheless be used for shape-finding). For approximating effects due to large displacements see section 6.5.2.

Displacements can only be prescribed if the corresponding displacement degree of freedom is removed from the statical system. This means you have to activate the corresponding button in the Conditions-section of the PreDisp-component. The first three buttons stand for translations the last three for rotations. Only those components of the "Trans"- and "Rot"-vectors take effect which correspond to activated supports.

6.5. Algorithms

6.5.1. Analyze

With geometry, supports and loads defined the statical model is ready for processing. The Analysis-component computes the deflection for each load case and adds this information to the model. Whenever the Analysis-component reports an error (turns red) despite the fact that the Assemble component works, it is probably a good idea to check the support conditions.

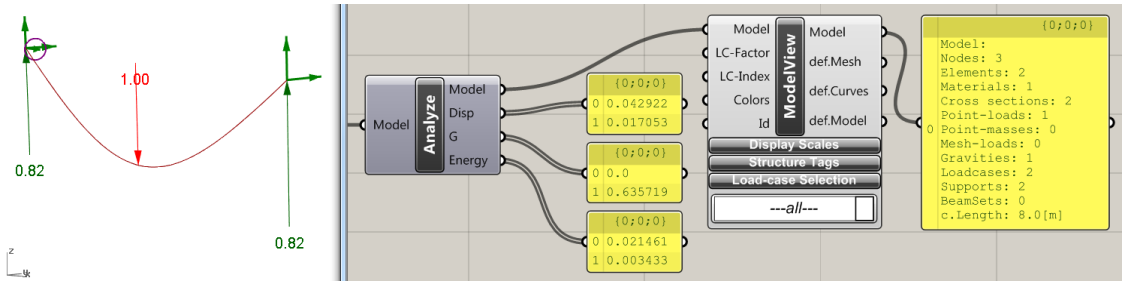


Figure 52: Deflection of simply supported beam under single load in mid-span and gravity.

Figure 52 shows a deflected beam. The analysis component not only computes the model deflections but also outputs the maximum nodal displacement (in meter), the maximum total force of gravity (in kilo Newton) and the structures internal deformation energy of each load case - see section 6.6.1 for details on work and energy.

These values can be used to rank structures in the course of a structural

optimization procedure: the more efficient a structure the smaller the maximum deflection, the amount of material used and the value of the internal elastic energy. Real structures are designed in such a way that their deflection does not impair their usability. See section A.3 for further details. Maximum deflection and elastic energy both provide a benchmark for structural stiffness yet from different points of view: the value of elastic energy allows to judge a structure as a whole; The maximum displacement returns a local peak value.

In order to view the deflected model use the ModelView-component (see section 6.6.2) and select the desired load case in the menu "Load case Selection". There exist two options for scaling the deflection output. First there is a slider entitled "Deformation" in the menu "Display Scales" that lets you do quick fine-tuning on the visual output. Second option: the input-plug "LC-Factor" which accepts a list of numbers that ModelView uses to scale the loads. Its default value is 1.0. Each item in the list applies to a load case. If the number of items in this list and the number of load cases do not match then the last number item is copied until there is a one to one correspondence. The second option for scaling displacements can be used in the course of form-finding operations: The model-plug at the right side of the ModelView outputs the displaced geometry which can be used for further processing. Selecting item "–all–" on the drop-down-list for the selected load case results in a superposition of all load cases with their corresponding scaling factor.

Looking at figure 52 one immediately notices that only beam center axes are shown. In order to see beams or shells in a rendered view add a "BeamView"- or "ShellView"-component after the ModelView. See sections 6.6.8 and 6.6.18 for details.

6.5.2. Analyze Large Deformation

Before the advent of digital modeling people like Heinz Isler or Antoni Gaudi helped themselves with physical models for generating curved geometries. A popular method was to use the shape of meshes or elastic membranes hanging from supports (see fig. 53).

In Karamba the behavior of hanging models can be simulated with the help of the "Analyze Large Deformation"-component. Figure 54 shows a geometry derived from an initially flat mesh under evenly distributed point-loads. Karamba handles geometric non-linearity by an incremental approach:

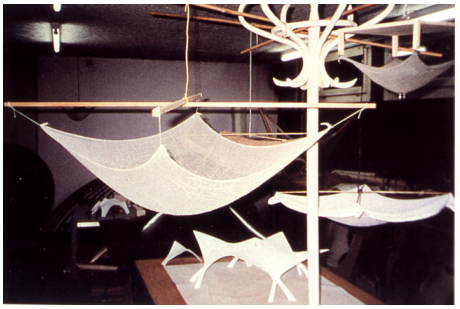
(a) (b) 

Figure 53: Hanging models. *Left:* Model of Antoni Gaudi for the Temple Expiatori de la Sagrada Família (from the internet). *Right:* Some of Heinz Islers hanging models (from the internet).

All external loads get applied in steps. After each step the model geometry updates to the deflected state. The more and the smaller the steps the better the approximation of geometric non-linearity. The purely incremental method however incurs an unavoidable drift from the exact solution. For form-finding this error should be negligible in most cases.

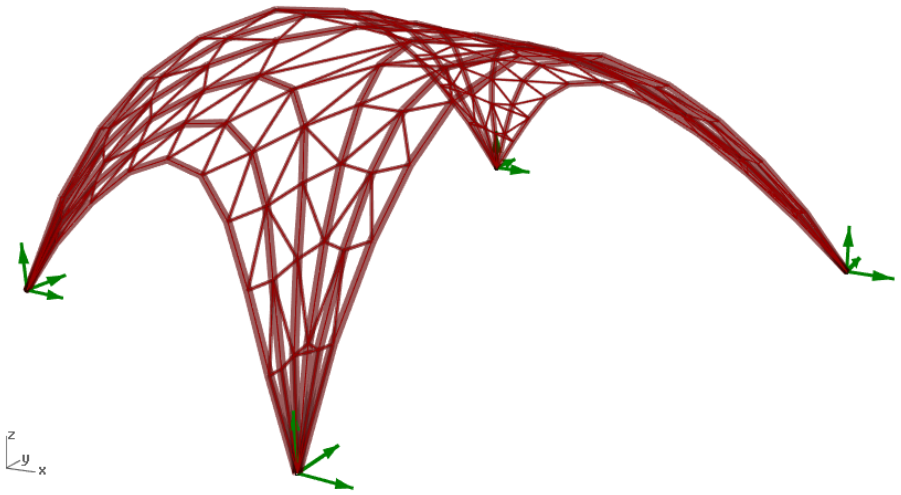


Figure 54: Structure resulting from large deflection analysis with the "LaDeform"-component.

Figure 55 shows a simply supported beam under the action of uniformly distributed point loads. Due to its slenderness axial stiffness by far outweighs bending stiffness. Thus the deflected shape corresponds to a string under self weight.

The "LaDeform" component has four input-plugs:

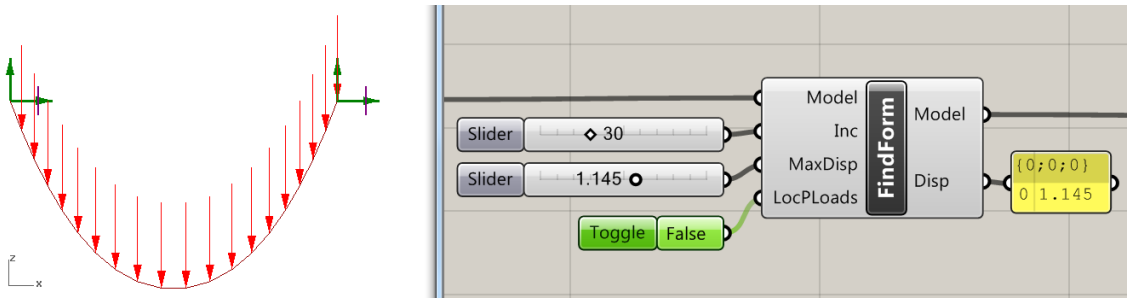


Figure 55: Catenary resulting from point loads that do not change their direction when displaced.

"Model" : model to be deformed. "LaDeform" uses load-case "0" for calculating the deflected shape.

"Inc" : number of increments for applying the loads

"MaxDisp" : maximum displacement to be reached in meter $[m]$. When supplied with a value the incremental deflection in each step is scaled to $MaxDisp/Inc$. This enables Karamba to handle problems with overly large deflections at the beginning of the incremental procedure. Think of an initially straight string: Due to its negligible bending stiffness it tends to deform tremendously in the first loading step.

With no value supplied in "MaxDisp" external loads get incremented proportionally in each step. Aside from cases like mentioned above this results in an approximation of the structures "real" deflections under the given loads.

"LocPLoads" : When "false" (the default) point-loads keep their initial direction. Fig. 56 shows what happens if "LocPLoads" is set to "true": The point-loads co-rotate with the points they apply to. This leads to a pneumatic shape. The direction of line-loads is not affected by "LocPLoads". Their orientation during form-finding corresponds to that given in the "Line-Load on Element"-component.

The two output plugs of the "LaDeform"-component supply the deflected model and the maximum deflection reached in the calculation.

The local coordinate system of each element gets updated along with its positions. By default an elements local Y-axis is taken parallel to the global X-Y-plane. If an element reaches a vertical position however, its default coordinate system flips – the Y-axis is then taken parallel to the global

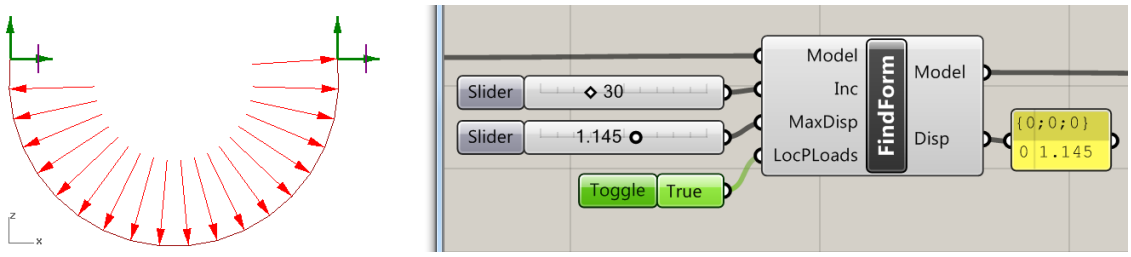


Figure 56: Pneumatic form resulting from point loads that rotate along with the points they apply to.

Y-axis. This may lead to unwanted results when using line-loads which flip along with the local coordinate system. It is possible to avoid this by defining local axes via the "OrientateBeam"-component.

In each incremental step the internal forces of the previous step get cleared. This is the reason why the resulting, deflected model contains no information regarding internal forces.

6.5.3. Eigen Modes

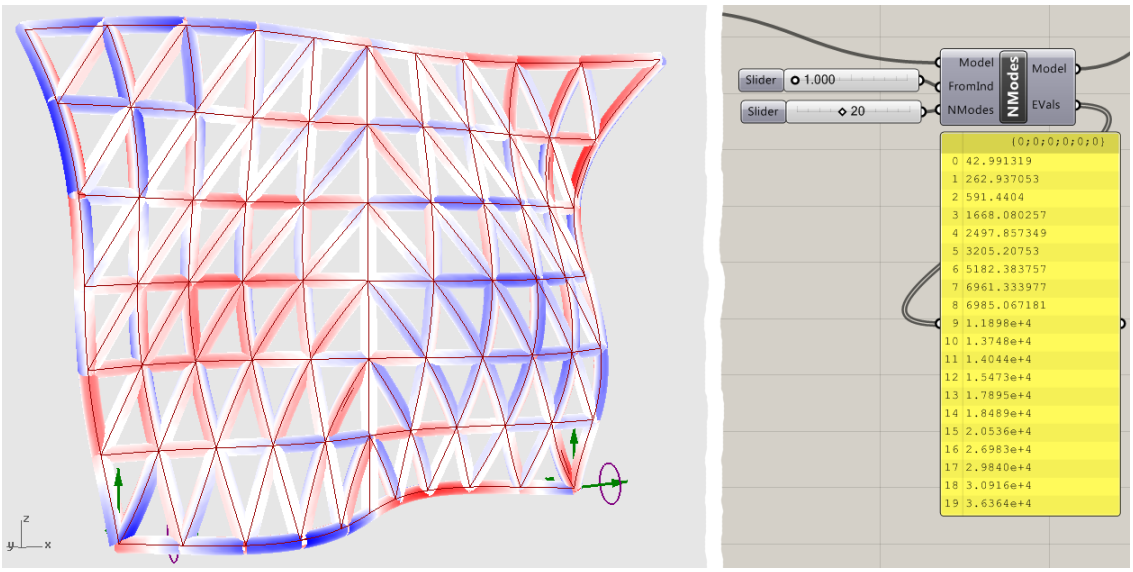


Figure 57: Left: 14^{th} eigen-mode with strain display enabled. Right: EigenMode-component in action.

Karambas EigenMode-component allows to calculate eigen-modes and corresponding eigen-values of structures (see figure 57) as used for buckling analysis i.e. without inertia effects. For details on eigen-modes and natural-

modes see section A.5).

The input parameters are a model, the index of the first eigen-mode to be computed and the number of desired eigen-modes. The model which comes out on the right side lists the computed eigen-modes as load cases. Thus they can be superimposed using the ModelView-component for form-finding or structural optimization. All loads which were defined on the input model get discarded. The determination of eigen-shapes can take some while in case of large structures or many modes to be calculated. **Grasshopper has no "Cancel"-button. Therefore you should save your model before activating the component!**

The number of different eigen-modes in a structure equals the number of degrees of freedom. In case of beams there are six degrees of freedom per node, with only trusses attached a node possesses three degrees of freedom. Figure 58 shows the first nine eigen-modes of a triangular beam mesh that is fixed at its lower corners. In the upper left corner of figure 58 one sees the undeflected shape. The higher the index of an eigen-mode the more folds it exhibits.

The eigen-values represent a measure for the resistance of a structure against being deformed to the corresponding eigen-form. Values of zero or nearly zero signal rigid body modes.

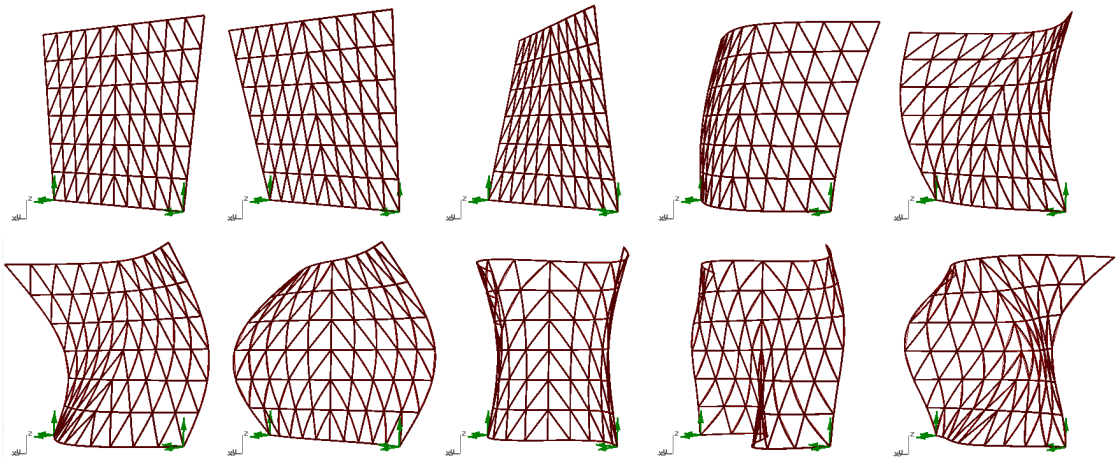


Figure 58: Undeflected geometry (upper left corner) and the first nine eigen-modes of the structure.

6.5.4. Natural Vibrations

In case you want to know how and at which frequency a structure vibrates use the "NaturalVibrations"-component (see fig. 59). For details on eigen-modes and natural-modes see section A.5).

The mass of beams and trusses enters with the values derived from their material weight. Karamba uses consistent mass matrixes for beam elements. For truss and shell elements a lumped approach is applied.

At nodes additional masses (see sec. 50) can be defined to simulate the effect of e.g. concrete slabs (these normally make up the majority of mass in high-rises) in an approximate manner. These masses are assumed to have translational inertia only.

Karamba scales the resulting vibration modes $\vec{v_i}$ in such a way that their largest component is one. They get attached to a model as load-cases which can be viewed via a "ModelView"-component. The calculation of modal mass and participation factors are based on the modal displacements as scaled in the above described manner.

The calculation of natural vibrations is computationally more demanding than eigen-modes. **So save your model before activating the component! Grasshopper has no "Cancel"-button.**

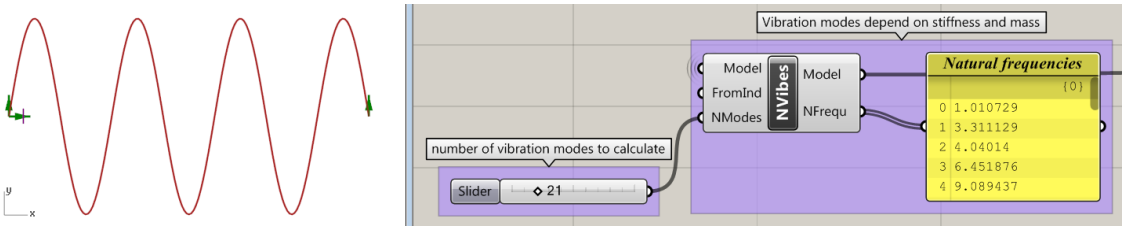


Figure 59: Simply supported steel beam IPE100 of length $10[m]$ in its 14^{th} natural vibration mode.

6.5.5. Force Flow Finder

Evolutionary structural optimization (ESO) constitutes a method of topology optimization which was pioneered by Y.M. Xie and G.P. Steven. The underlying principle is simple: One starts from a given volume made up from structural elements on predefined supports and with preset loads acting on it. Calculating the structural response will show that there are regions which carry more of the external load than others. Now one removes a number of those elements of the structure that are least strained and thus least ef-

fective. Again the response of the now thinned out model is determined, under-utilized elements removed and so on. This iterative procedure stops when a target volume or number of remaining structural elements is reached.

The above algorithm can be viewed as a way of tracing the internal force flow through a structure and removing those elements that do not form part of it. That is what the "ForceFlowFinder"(FFF)-component does (see fig. 60 of a cantilever after applying FFF). Yet its main emphasis does not lie on structural optimization, but on harnessing the flow of forces for design purposes. The "ForceFlowFinder"-component replaces the "ESO"- and "BESO"-components of previous Karamba releases. It works on beam and truss elements only. Shells are currently not included in the algorithm.

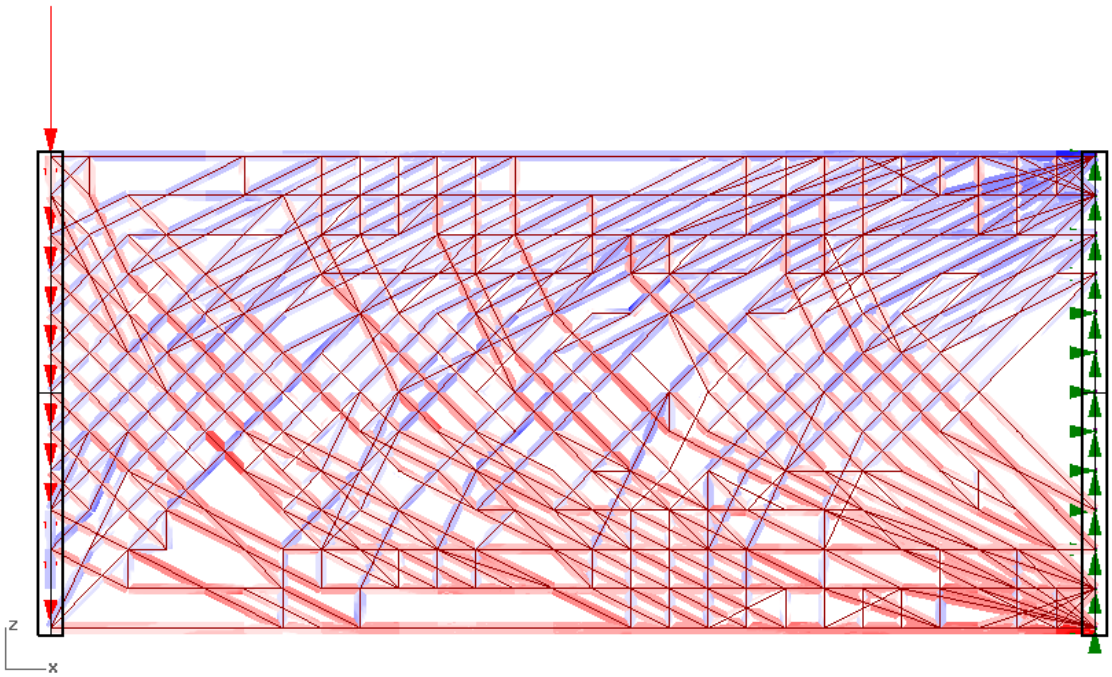


Figure 60: Cantilever with initially regular mesh after application of the "ForceFlowFinder"-component.

Figure 61 shows the "ForceFlowFinder"-component at work. On the left side one can see the initial geometry which is a triangular mesh derived from a surface. There exist two load cases with loads acting in the plane of the structure in horizontal and vertical direction respectively. Three corner nodes of the structure are held fixed. The right picture shows the optimized structure reduced to 45% of its initial mass in the course of 20 design iterations.

Here the description of the input parameters:

"Model" : receives the model to be processed.

"Elem Id" : There are three alternatives concerning this input parameter:

- No input: The whole of the structure will be considered by the FFF.
- The input consists of one string: All elements whose identifiers match take part.
- A list of strings is given: Elements that match a given list entry belong to one group. They get collectively activated or deactivated during force path finding. A structure may consist of active and non-active elements. The initial state of a group is determined by the state of the majority of its elements. Groups need not be disjoint.

"LCase" : List of load cases to be considered. Zero is the index of the first load case. Considering the total effect of several load cases amounts to adding up their individual influences on an element.

"Target" : ratio of the target mass to the initial mass of a structure. When determining the initial mass all elements of the structure – irrespective of state of activation – count. In the target structure only active elements contribute to its mass. This enables one to apply FFF-components in series. Depending on the activation status of the model elements applying FFF will lead to an increase or decrease in the number of active elements. The activation status of individual elements can be set by means of the "ModifyBeam"- and "ActivateModel" components.

"Iter" : Number of iterations within which the target mass of the structure should be reached. Is limited by "MaxIter". If the number of iterations is selected too low then it may occur that single beams get disconnected from the main structure and they seem to fly. The reason for this lies in the fact that Karamba applies a so called soft-kill approach for thinning out the structure: elements are not removed but simply given small stiffness values. This ensures that structural response can be calculated under all circumstances.

"MaxIter" : Maximum number of iterations. When set to a value larger than "Iter" then at least one element will change its activation state during

the last $MaxIter - Iter$ iterations. Sometimes this helps to improve results.

"Factors for weighting forces/moments" : The FFF-component lets you select weighting factors for the different force and bending components in an element. The weight of an element is determined by averaging single force-components at its end, division by the elements mass and multiplication by the corresponding user given weighting factor. The weight of groups results from the average of their members. These are the available weighting factors:

- "WTension": factor for axial tension force
- "WCompr.": factor for axial compression force
- "WShear": factor for resultant shear force
- "WMoment": factor for resultant moments

"Overdrive" : Say in each iteration step there needs to be a mass of $n[kg]$ removed in order to meet the structures target mass in the given $Iter$ number of iterations. With $Overdrive = m$ there will be $(m+1) \cdot n$ active elements moved to the pool of inactive elements. An evaluation of the structures response follows. In a second step $m \cdot n$ members get flipped from inactive to active so that the balance is right again. This adds a bi-directional component to the FFF-process which often leads to improved results.

"MinDist" : In some cases one wishes to limit the number of elements that get added or removed in a certain area. "MinDist" lets you select the minimum distance in meter $[m]$ between the endpoints of elements that may be changed in one iteration.

"WLimit" : At the end of the FFF-process it often occurs that a small fraction of the elements is much less utilized than the average. "WLimit" lets you remove those elements whose weight is below "WLimit" times the average weight of elements.

On the right side of the "ForceFlowFinder"-component these output-plugs exist:

"max.disp" : maximum displacement of the resulting model from among all load cases.

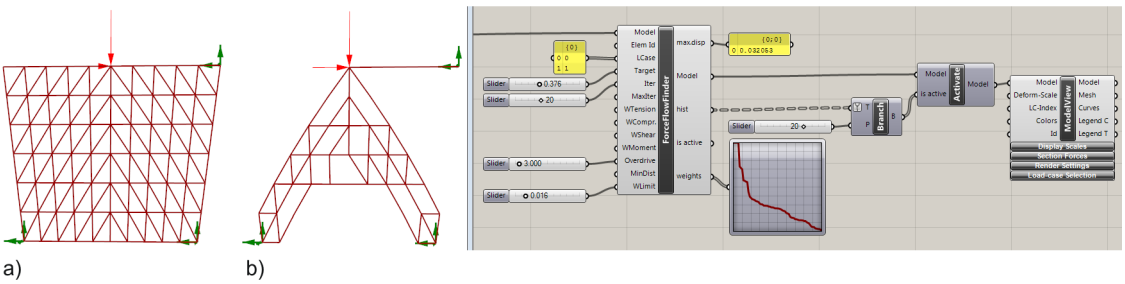


Figure 61: Triangular mesh of beams before (a) and after (b) applying the "FindForcePath"-component.

"Model" : structure with element activation according to the force path found.

"hist" : a data tree which contains for each iteration step a list of boolean values that signify whether an element in active (true) or inactive (false). The boolean values map directly on the model elements. Using a "Tree Branch" component with a slider connected to a "Activate Model"-component (see section 6.1.1) lets you inspect the history of the FFF-process (see fig. 61).

"is active" : renders a list of true/false values – one for each element. True signals that the corresponding element is part of the final structure (i.e. active). Otherwise it contains a "false" entry.

"weights" : List of element or group weights in ascending order in the final structure. This can be used as a qualitative check of the result: The more evenly distributed the weights, the better utilized the structure. There will always be force concentrations around supports and external loads which show up as sharp peaks. A good way of visualization is to use a "Quick Graph"-component (see fig. 61).

6.5.6. Tension/Compression Eliminator

The "Tension/Compression Eliminator"-component removes elements from a model based on the sign of their axial force.

These are the available input parameters:

"Iter" The removal of tensile or compressive elements works in an iterative fashion. The procedure stops either when no changes occur from one

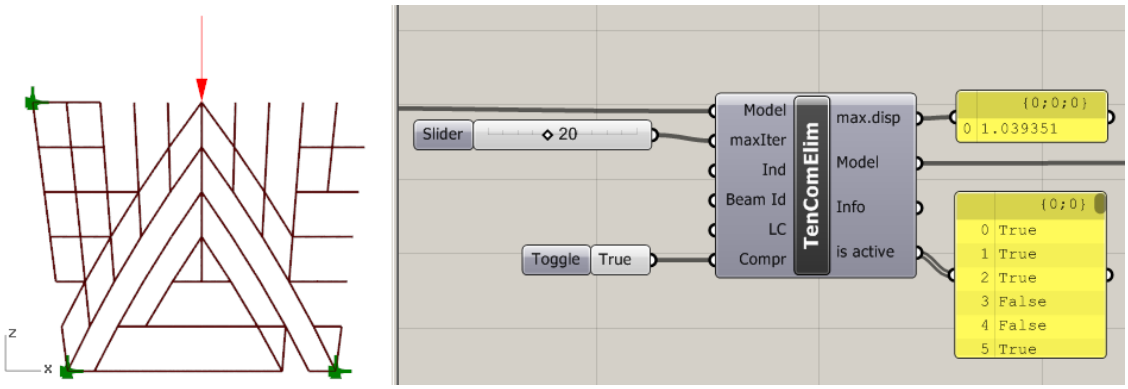


Figure 62: The "Tension/Compression Eliminator"-component.

step to another or if the the maximum number of iterations "Iter" is reached.

"Ind" Indexes of the elements that may be removed in the course of the procedure. By default the whole structure is included.

"LC" You can specify a special load case to consider. The default is "0".

"Compr" If true, then only members under compression will be kept. Otherwise only members under tension will survive. This value is false by default.

Elements selected for removal are assigned a negligible stiffness (i.e. a soft-kill approach is used).

6.5.7. Optimize Cross Section

Use the "Optimize Cross Section"(OptiCroSec)-component for the automatic selection of the most appropriate cross sections of beams and shells. It takes into account the cross sections load bearing capacity and optionally limits the maximum deflection of the structure.

Figure 63 shows a typical set-up. The initial structure consisted of I-sections of type HEA100 which have a height and width of $100[mm]$. They could not sustain the given load: The resulting bending stresses would lie way beyond the yield stress of the assumed material which is steel S235 with $f_y = 23.5[kN/cm^2]$.

First the OptiCroSec-component determines the cross section of each element in such a way that their load-bearing capacity is sufficient for all load-cases. In order to achieve this, Karamba uses the following procedure:

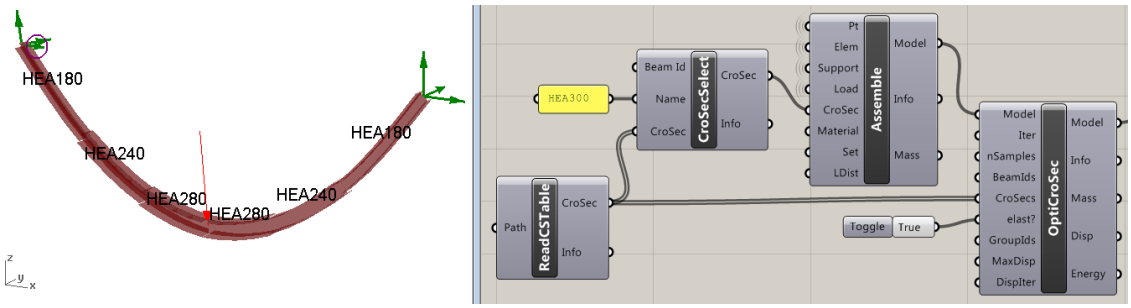


Figure 63: Cross section optimization with the OptiCroSec-component on a simply supported beam.

1. Determination of section forces at "nSample" points along all beams using the initial cross section
2. For each element or given set of elements: selection of the first sufficient entry from the family to which each cross section belongs
3. If no changes were necessary in step two or the maximum number of design iterations is reached, the algorithm stops. Otherwise it returns to step one using the cross sections selected in step two.

In statically indeterminate structures the section forces depend on the stiffness (i.e. cross section) of the members. This necessitates the iterative procedure described above.

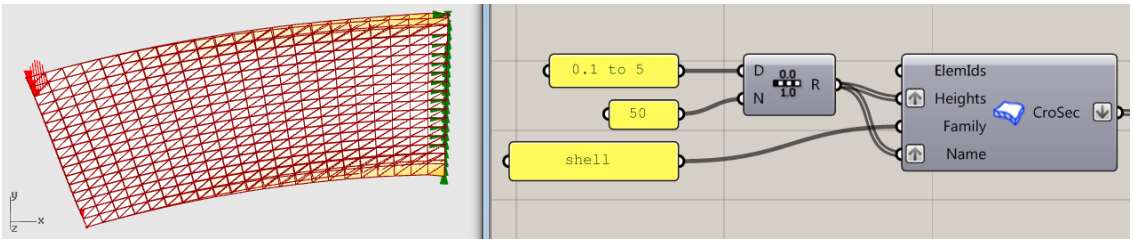


Figure 64: Cross section optimization with the OptiCroSec-component on a cantilever discretized with shell elements.

When performing thickness optimization on shells the list structure for defining a cross section family plays a crucial role. In fig. 64 one can see the example of a cantilever: The optimization results in thicker shells at the top and bottom edge of the built-in side. When defining a list of shell cross sections one needs to supply a list of lists of thicknesses and corresponding cross section names. Each sub-list specifies the element thicknesses of one shell (see section 6.2.2).

After ensuring safety against structural failure a second, optional step follows where Karamba tries to approach a user supplied maximum deflection. Behind the scenes Karamba iteratively adapts temporarily the yield stress of the materials. This may lead to uneconomic results in case of structures where the maximum displacement occurs in a small region whereas the rest of the structure shows a much smaller deformation. In order that the iterative adaption for the maximum displacement works, the number of design iterations should be chosen appropriately – five is normally sufficient.

Building codes prescribe different levels of safety against reaching maximum displacement and load bearing limits. When using external loads on ultimate limit state level one should keep in mind that this is approximately 1.4 times the loads used to check maximum displacement requirements.

When the given loads surpass the load bearing capacity of the biggest cross section available in a cross section family, Karamba issues a warning via the "Info" output-plug.

There is no guarantee, that the iteration procedure for finding the optimal cross sections eventually converges – so check the results via the utilization-output of the "ModelView"-component. Due to the lower-bound theorem of plasticity the structure will be sufficient for the given loads at any iteration step – although some elements may show overutilization – provided that the material is sufficiently plastic (like e.g. steel). With increasing number of iterations the statical system tends to become more and more statically determinate.

The profile selection procedure assumes that the cross sections of a family are ordered: starting with your most favorite and descending to the least desired cross section. In the cross section table "CrossSectionValues.csv" that comes with Karamba all families are ranked according to their height. The cross section with the smallest height comes first, the one with the largest height last. When using cross section area as sorting criteria, structures of minimum weight (and thus approximately cost) result. See 6.2.11 for how to switch between minimum height and minimum weight design. Ordering the profiles by area may lead to structures where the cross section heights vary significantly from one beam to the next.

In order to check whether a given beam cross section is sufficient Karamba applies a procedure that approximates that for steel according to Eurocode 1993-1-1. It takes account of the normal force, biaxial bending, torsion and shear force. For more details see section A.6. It is possible to switch off the influence of buckling for single members or set user defined values for the

buckling length (see section 6.1.11).

The adverse effect of compressive normal forces in a beam can be taken into account. In order to do this one needs to determine the buckling length l_b of an element. For this the following simplification – which is not always on the safe side – is applied: Starting from the endpoints of an element, proceeding to its neighbors, the first nodes are tracked that connect to more than two elements. The buckling length is determined as the distance between these two nodes. It lies on the safe side in case of endpoints held by the rest of the structure against translation. When beams are elements of a larger part of a system that buckles (e.g. a girder of a truss) then the applied determination of buckling length produces unsafe results! In case of a free end the buckling length is doubled. Compressive normal force in slender beams reduces their allowable maximum stress below the yield limit. Visualizing the level of utilization with the ModelView-component will then show values below 100% in the compressive range.

The OptiCroSec-component provides the following set of input-plugs:

"Model" Model to be optimized

"Iter" Maximum number of design iterations for sufficient load bearing capacity. The default value is five.

"nSamples" Number of points along beams at which their utilization is determined. The default is three.

"ElemIds" Identifiers of elements that should be optimized. If not specified, optimization is carried out for the entire model.

"CroSec" Cross section-list that contains families of cross sections ordered from most favorite to least desired. Family membership of cross sections is given via their "family" property.

"elast?" If set to "true" (the default) cross section design is done within the elastic range. This means that under given loads the maximum resulting stress in a cross section has to lie below the yield stress f_y of the material. In case of materials with high ductility (like steel) the plastic capacity of cross sections can be exploited. Depending on the cross section shape the plastic capacity is 10% to 20% higher than the elastic capacity. Set "elast?" to "false" in order to activate plastic cross section design. When enabling plastic cross section design do not

be surprised that the ModelView reports utilization-levels beyond 100%. The reason is that Karamba assumes linear elastic material behavior.

"GroupIds" Sometimes constructive logic demands that all elements of a given set have identical cross sections. Karamba allows to define beam sets (see section 6.1.10). Put the set identifiers into the "GroupIds"-plug to enforce identical cross sections.

"MaxDisp" For usability of a structure it is necessary to put a limit on its maximum deflection. This can be done using the "MaxDisp"-plug. By default its value is "-1" which means that the maximum deflection is not considered for cross section design.

"DispIter" Maximum number of iterations used to reach the maximum displacement criteria in case there is one. The design iterations for maximum displacement come after those for load bearing capacity.

On the output side the "Model"-plug renders the structure with optimized cross sections. Check the "Info"-plug in order to see whether any problems occurred during optimization. The "Mass"-plug informs you about the overall mass of the optimized structure. "Disp"- and "Energy"-plugs return the maximum displacement and internal energy of the structure after the last cross section design iteration.

Nonstandard cross sections, e.g. those that you custom design, may show local buckling. This means they loose their load-bearing capacity either before the maximum stress reaches the yield limit (in case of elastic design) or before they form a yield hinge (in case of plastic design). Slender I-sections or hollow sections with very thin wall thickness may be affected. Karamba does not take account of that. Consult you local building code on this.

The aim of the design procedure applied in Karamba is to render plausible cross section choices. It builds upon a range of simplifications (e.g. determination of buckling length, interaction of multiple section forces) and omits some phenomena that might be decisive (e.g. local buckling of slender sections, lateral torsional buckling, system buckling).

6.6. Results

The results category consists of three sections. The first contains components that apply to a structure in general. Components of the second and third category apply to beams and shells respectively.

6.6.1. Deformation-Energy

In mechanics energy is equal to force times displacement parallel to its direction. Think of a rubber band: If you stretch it you do work on it. This work gets stored inside the rubber and can be transformed into other kinds of energy. You may for example launch a small toy airplane with it: Then the elastic energy in the rubber gets transformed into kinetic energy. When stretching an elastic material the force to be applied at the beginning is zero and then grows proportionally to the stiffness and the increase of length of the material. The mechanical work is equal to the area beneath the curve that results from drawing the magnitude of the applied force over its corresponding displacement. In case of linear elastic materials this gives a rectangular triangle with the final displacement forming one leg and the final force being its other leg. From this one can see, that for equal final forces the elastic energy stored in a material decreases with decreasing displacements which corresponds to increasing stiffness.

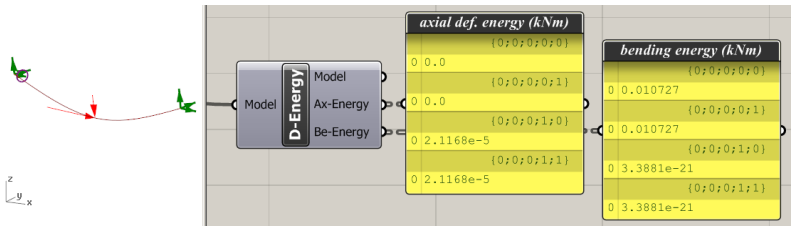


Figure 65: Simply supported beam under axial and transversal point-load: List of axial deformation energy and bending energy for each element and load case.

The structure of the data tree returned from the "D-Energy"-component (see fig. 65) corresponds to Model/Load-case/Element-result. In case of shells the list on the lowest level contains the axial or bending energy of each element of the shell.

6.6.2. ModelView

The ModelView-component of the "Results" subsection controls the general display properties of the statical model (see figure 14). More specific visual properties that relate to beam and shell elements can be defined with the "BeamView" and "ShellView"-component. The viewing options get stored in the model. Settings of View-components thus stick with the model and remain valid further down the data-stream until changed by another View-component.

When adding a ModelView to the definition it is sometimes a good idea to turn off the preview of all other components so that they do not interfere. Clicking on the black menu headings unfolds the ModelView and unveils widgets for tuning the model display. Each of these will be explained further below. The range and current value of the sliders may be set by double-clicking on the knob.

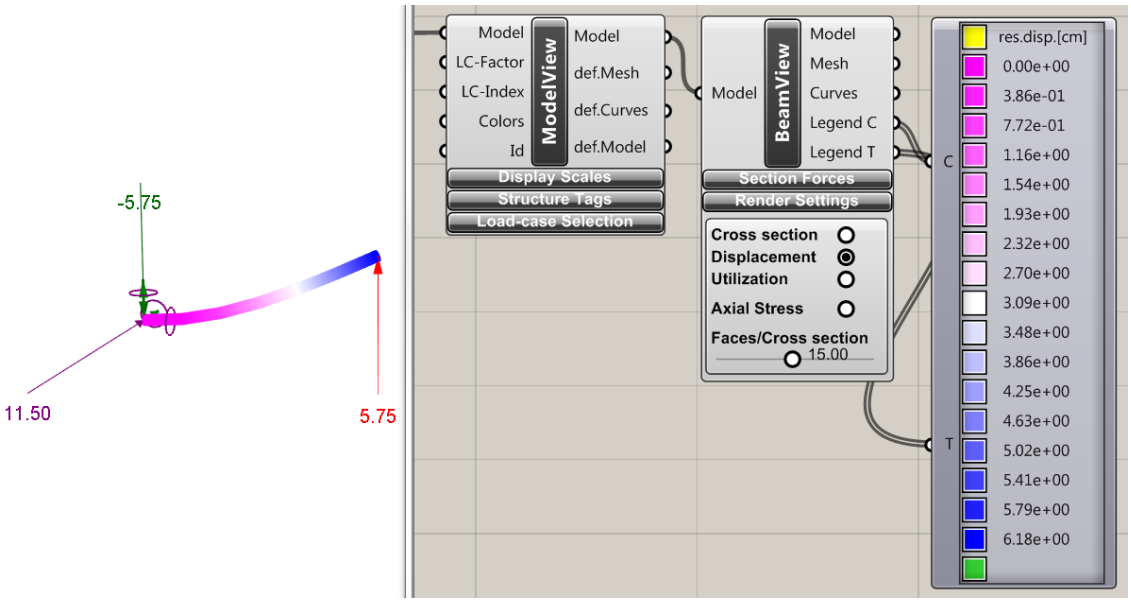


Figure 66: Partial view of a model.

The ModelView-component features five plugs on its left side:

- "Model" expects the model to be displayed
- "LC-Factor" can be used to scale individual load cases (see further below).
- "LC-Index" lets one select the visible load-case. The value in "LC-Index" will be added to the load-case selected in the drop-down-list of ModelView ("—all—" counts as -1). If the resulting number is larger than the number of available load-cases the ModelView turns red. If the resulting value is smaller than 0 all load-cases are superimposed. The possibility of using a number-slider for selecting load-cases makes life easier in case that there are many of them.
- "Colors": Color plots for e.g. stresses use a color spectrum from blue to white to red by default. One can customize the color range by handing

over a list of RGB-values to the "Colors"-plug. There have to be at least four colors given. The first color is used for values below, the last color for values above the current number range. The remaining colors get evenly distributed over the number range. The Grasshopper component "Gradient" can be used to generate the list of colors (see fig. 67). In case you want to change the coloring defaults, set them in the "karamba.ini" file.

- "Id": This plug lets one select those parts of a model which shall be displayed. It expects a list of strings. The default value is an empty string which means that all of the model shall be visible. As one can see in fig. 66 it is possible to input regular expressions. These must start with the character "&" and adhere to the conventions for regular expressions as used in C#. The identifier of each element of the model is compared to each item of the given string list. In case a list entry matches the element identifier the element will be displayed.

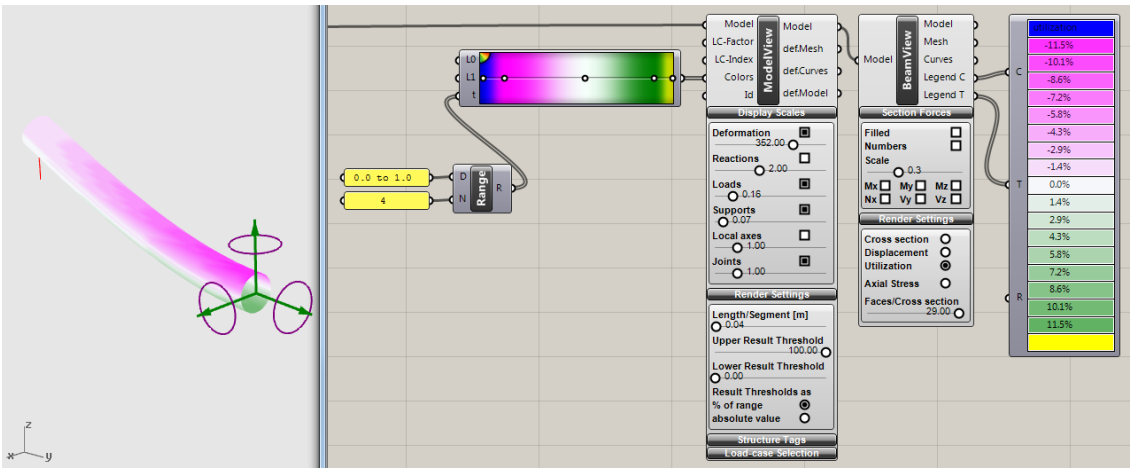


Figure 67: Color plot of strains with custom color range.

There are five output plugs on the ModelView-component:

- "Model" is the model which was fed in on the left side.
- From the "def.Mesh" output-plug you can get the mesh of the shells of the deformed model for further processing. It is a list of meshes with each item corresponding to one shell.
- The "def.Curve" plug delivers the axes of the beams of the deformed

structure as interpolated 3^{rd} degree nurb-splines. Use the Length/Subdivision slider to set the number of interpolation points.

- "def.Model": When there are results available from a statical calculation deflections are scaled and added to the node coordinates of the original model so that it contains the deformed geometry.

The "Display Scales"-submenu

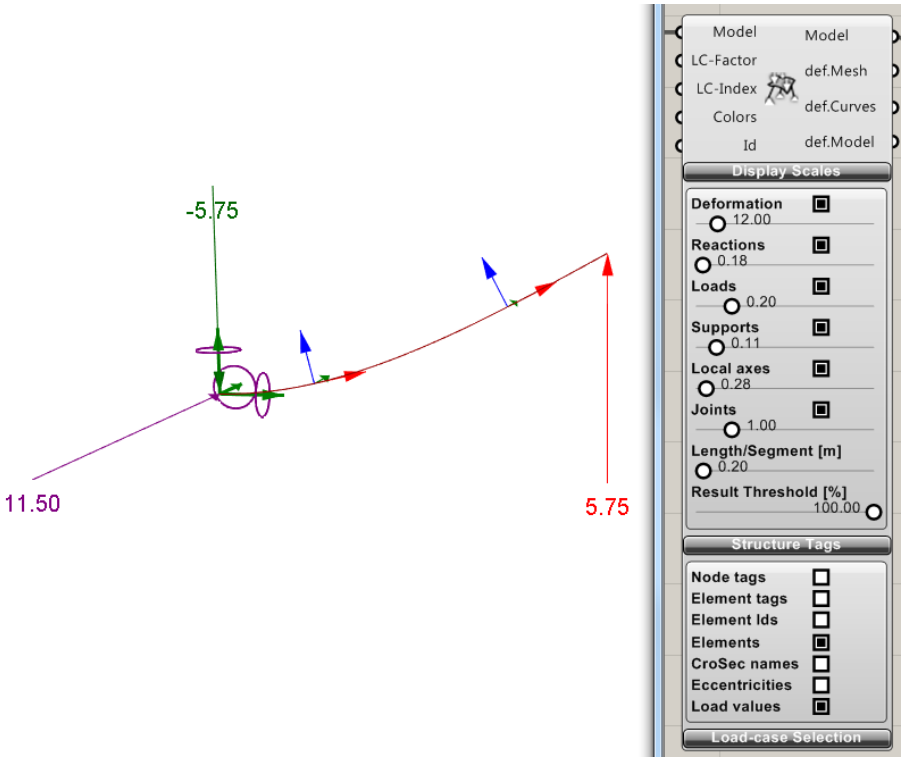


Figure 68: Local axes of cantilever composed of two beam elements, reaction force and moment at support.

The "Display Scales"-submenu contains check boxes and sliders to enable/disable and scale displacements, reaction forces at supports, load-symbols, support-symbols, local coordinate systems and symbols for joints at the endpoints of elements. The displacement scale influences the display and the output at the model-plug. It has no effect on stresses, strains, etc.. The colors of the local coordinate axes red, green, blue symbolize the local X-, Y-, and Z-axis.

The "Render Settings"-submenu

The slider entitled "Length/Segment[m]" lets one control the distance at which beam results (displacements, forces, moments, etc.) are plotted. It

also sets the number of lines that come out of the "def.Curves"-output-plug.

In some cases the color display of results gets distorted by the presence of stresse concentrations or utilization peeks. They make much of the structure look unstrained with some small patches of color where the peeks are. The "Upper Result Threshold"- and "Lower Result Threshold"-sliders let you filter out these extreme values. In case of the "Upper Result Threshold"-slider a value of x% sets the upper boundary value of the color range in such a way that x% of the actual value range is below. For the lower threshold it is vice versa. Values in the model beyond the given thresholds are given special colors to make them easily recognizable.

By default the result threshold values given above refer to the value range in percent. Sometimes it turns out to be practical to prescribe absolute values as thresholds (e.g. the yield stress of a material). The radio button group "Result Threshold as" can be used to switch between relative and absolute thresholds.

Limiting the value range of utilization values can be confusing: If the result thresholds are given in percent, then setting the lower threshold to zero and the upper to 100 displays the full range of utilization values. If the result thresholds are given as absolute values then a lower threshold of -100 and an upper threshold of 100 limit the color range to the areas where the material resistance is sufficient.

The "Structure Tags"-submenu

The "Structure Tags" menu contains checkboxes for adding visual information to parts of the model:

- "Node tags" attaches node-indexes to each node.
- "Element tags" attaches element-indexes to each element.
- "Element Ids" displays the element identifiers.
- "Elements": If enabled the "def.Curves" output-plug emits the axis of the deformed elements as lines and shows them on the Rhino-canvas.
- "CroSec names" displays the name of the cross-section of each element
- "Material names" displays the name of the material of each element
- "Eccentricities" visualizes beam eccentricities as blue lines at the end-points if active.

- "Load values" adds the numerical values of loads or point masses to the corresponding sysmbols.

The "Load-case Selection"-submenu

The "Load-case Selection" menu contains a drop-down list from which one can choose the load-case which should be displayed. By default it is set to "—all—" which means that the results of all load-cases are superimposed. Define load-factors by feeding a corresponding list of numbers into the "LC-Factor" input-plug.

The "Load-case Selection" sets the load-case to be queried for the shell results-components placed further downstream (e.g. "Force Flow Lines on Shells", "Principal Stress Lines on Shells",...).

6.6.3. Nodal Displacements

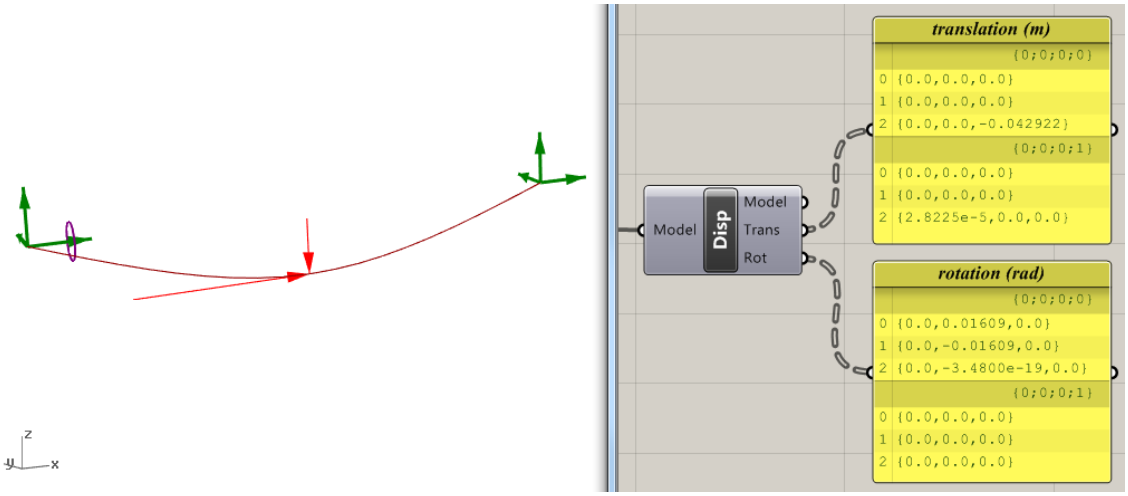


Figure 69: Simply supported beam under axial and transverse point-load: List of nodal displacements: vectors with translations and rotations for each node and load case.

The "NodeDisp" component lists the displacements of each node for all load cases. Two data-trees consisting of vectors make up its output. The two rightmost dimensions correspond to Model/LoadCase. The data for each node at the output plugs "Trans" and "Rot" consists of a vector which contains the three translations or three rotations (see fig. 69). The vectors refer to the global coordinate system. Their units are meter and radiant respectively. A positive rotation say about the global X-axis means that the node rotates counter clockwise for someone who looks at the origin of the coordinate system with the X-axis pointing towards him or her.

6.6.4. Approximate Principal Strains

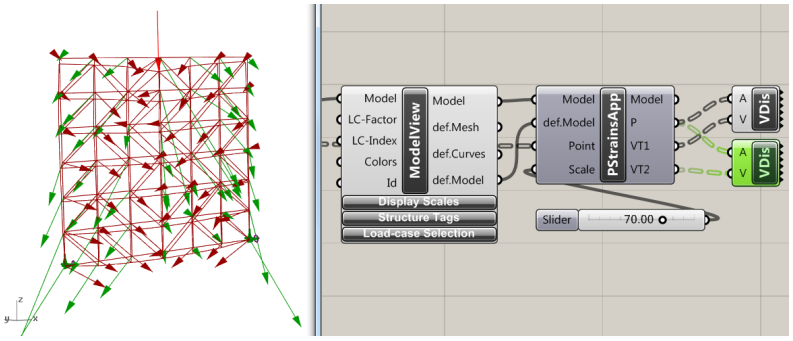


Figure 70: Approximation of principal strains in a simply supported slab simulated with beam elements under a point-load. Irregularity of principal strain directions is due to the irregularity of the element grid.

Karamba includes shell elements from which principal stress lines can be retrieved (see sec. 6.6.16). In case of single layer grid shells made up of beams the "Approximate Principal Strains"-component can be used to determine the approximate principal strain directions of such structures (see fig. 70). It works on arbitrary sets of deformed points.

The calculation of principal strains is based on the assumption of a continua. When applied to nodes connected with linear elements the result can thus only result in a qualitative picture – therefore the term "Approximate".

The "Approximate Principal Strains"-component expects as input a reference model (input-plug "Model") and the same model in a deformed configuration (input-plug "def.Model"). The deformed model can be the output of a ModelView-component. Hand over a list of points to the input-plug 'Point' where principal strain directions shall be computed. For each point in this list the following two steps are applied: First those three nodes of the reference model that do not lie on a line and have minimum distance to the given point are determined. Second the strains in the sides of the thus found triangle determine the principal strain directions – plane stress is assumed. The conversion of first (output-plug "VT1") and second principal strains (output-plug "VT2") to vectors occurs in such a way that they align with the average displacement of the triangle that defines the corresponding strain-state. The size of the vectors emanating from "VT1" and "VT2" can be scaled by providing a factor in the input-plug "Scale".

The principal strains are tangents to the principal stress lines of a structure. Use e.g. Daniel Hambleton's "SPM Vector Components" (see `http:`

//www.grasshopper3d.com/group/spmvectorcomponents) to retrieve these lines from the strain-vector-field.

6.6.5. Reaction Forces

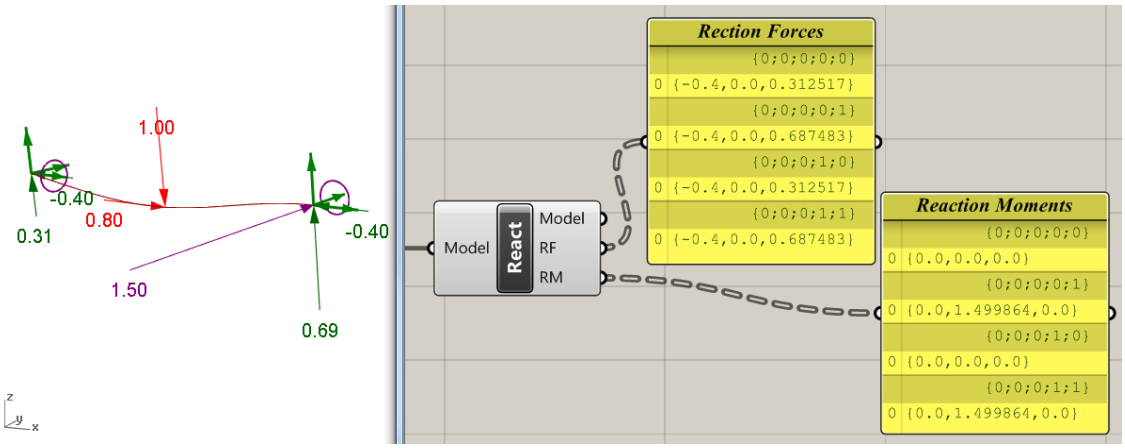


Figure 71: Beam under axial and transverse point-load: Reaction forces and moments for both load cases.

The "Reaction Forces"-component gives access to the reaction forces and moments at supports. It expects a model at its input-plug and returns via "RF" and "RM" a tree containing reaction forces in $[kN]$ and reaction moments in $[kNm]$ as three dimensional vectors. The two rightmost dimensions correspond to LoadCase/Support. The support reactions are ordered in such a way that the indexes of the nodes they attach to form an ascending sequence. In case of locally oriented supports, reaction forces refer to the local coordinate system.

6.6.6. Utilization of Elements

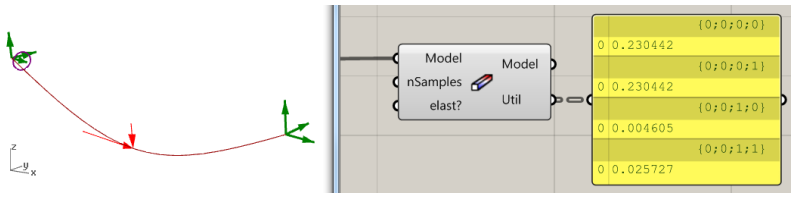


Figure 72: Simply supported beam under axial and transverse point-load: Utilization of the cross sections of the elements.

Use the "Utilization of Elements"-component in order to get the level of

utilization for each element in each load case (see fig. 72). The rightmost index is the load-case which comprises a list of one value per element. "1" means 100%. For beams the calculation approximates the procedure outlined in Eurocode 3 (see section A.6 for details). The utilization calculated for shells is the ratio between yield stress and Van Mises Stress in each element of the shell.

Utilization numbers for beams rendered by this component and the ModelView are different in case of compression: The ModelView-component returns the ratio of stress to yield stress as level of utilization, whereas the "Utilization of Elements"-component also includes buckling. See for example the last two entries on the bottom in fig. 72: The second load case is made up of an axial load acting in the middle of the beam. As both ends are axially fixed, one beam is in tension, on in compression. The absolute value of the normal force in both elements is the same. Yet the beam under compression has a utilization of 0.26, the one under tension only 0.05.

6.6.7. Beam Displacements

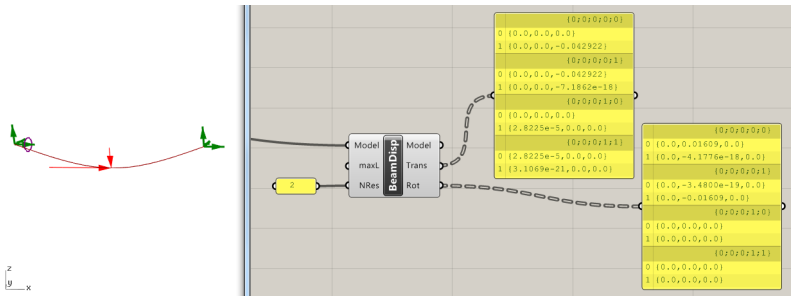


Figure 73: Simply supported beam consisting of two elements under axial and transverse point-load: List of displacements along the axis: three components of translations and rotations for each section and load case.

In case you want to know how displacements change over the length of a beam use the "Beam Displacements"-component (see fig. 73). The "maxL" and "'NRes" input-plugs work analogously to those of the "Section Forces"-component (see section 6.6.10).

6.6.8. BeamView

The "BeamView" components controls the display options related to beams (see fig. 74). This concerns the rendering of cross section forces, resultant

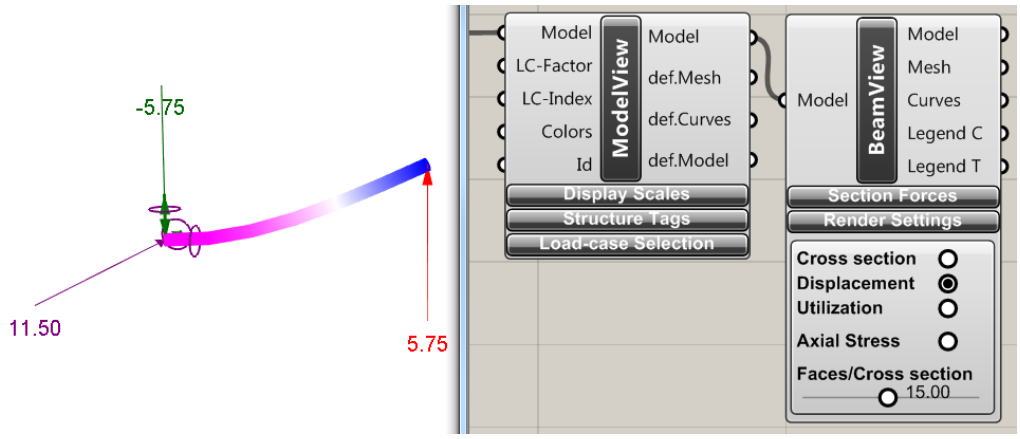


Figure 74: Display of resultant displacements on beam cross section.

displacements, utilization of material and axial stress.

The "Render Settings"-submenu

When activated, "Cross section", "Displacement", "Utilization" and "Axial Stress" result in a rendered view of the model. Utilization is calculated as the ratio between the normal stress at a point and the yield stress of the corresponding material. Shear and buckling are not considered.

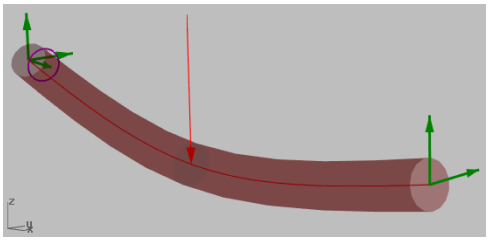
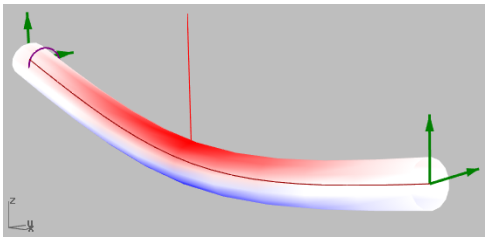
(a)  (b) 

Figure 75: Rendered images of the beam. *Left:* "Cross section"-option enabled. *Right:* "Axial Stress" enabled.

The color range of the results starts at the minimum value and stretches to the maximum. In case the model consists of one material, the zone of highest utilization will also be the zone of highest stress. Thus the distribution of colors will not change. You can define individual color ranges for all quantities in the "karamba.ini"-file. A "Legend"-component lets you inspect the meaning of the colors.

The mesh of the rendered image is available at the "Mesh"-output of the "BeamView"-component. Two sliders control the mesh-size of the rendered beams: First "Length/Segment" of "ModelView" determines the size of sections along the middle axis of the beams. Second "Faces/Cross section" of

"BeamView" controls the number of faces per cross-section.

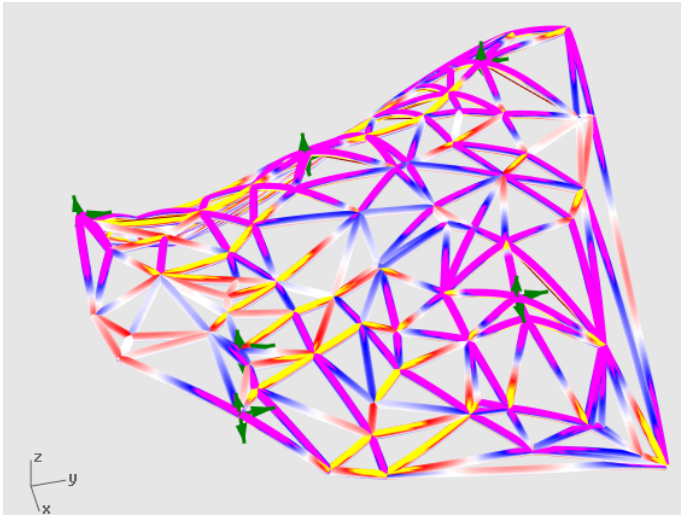


Figure 76: Mesh of beams under dead weight with Render Color Margin set to 5%.

It is instructive to see which parts of a beam are under tension or compression. Activate the "Stress"-checkbox in menu "Render Settings" in order to display the stresses in longitudinal beam direction. Red (like brick) means compression, blue (like steel) tension. In some models there may exist small regions with high stresses with the rest of the structure having comparatively low stress levels. This results in a stress rendering that is predominantly white and not very informative. With the sliders for "Result Threshold" of the "ModelView" you can set the percentage of maximum tensile and compressive stress at which the color-scale starts. Compressive stress values beyond that level appear yellow, excessive tensile strains green (see figure 76).

Display of cross section forces and moments

The "Section Forces" sub-menu lets you plot section forces and moments as curves, meshes and with or without values attached. All generated curves and meshes get appended to the BeamViews curve and Mesh output. The graphical representation is oriented according to the local coordinate axes of the beam and takes the undeflected geometry as its base. The index of bending moments indicates the local axis about which they rotate, for shear forces it is the direction in which they act (see also fig. 78). Customize the mesh-colors via "karamba.ini". The slider "Length/Subdivision" in sub-menu "Render Settings" of the ModelView-component controls the number of interpolation points.

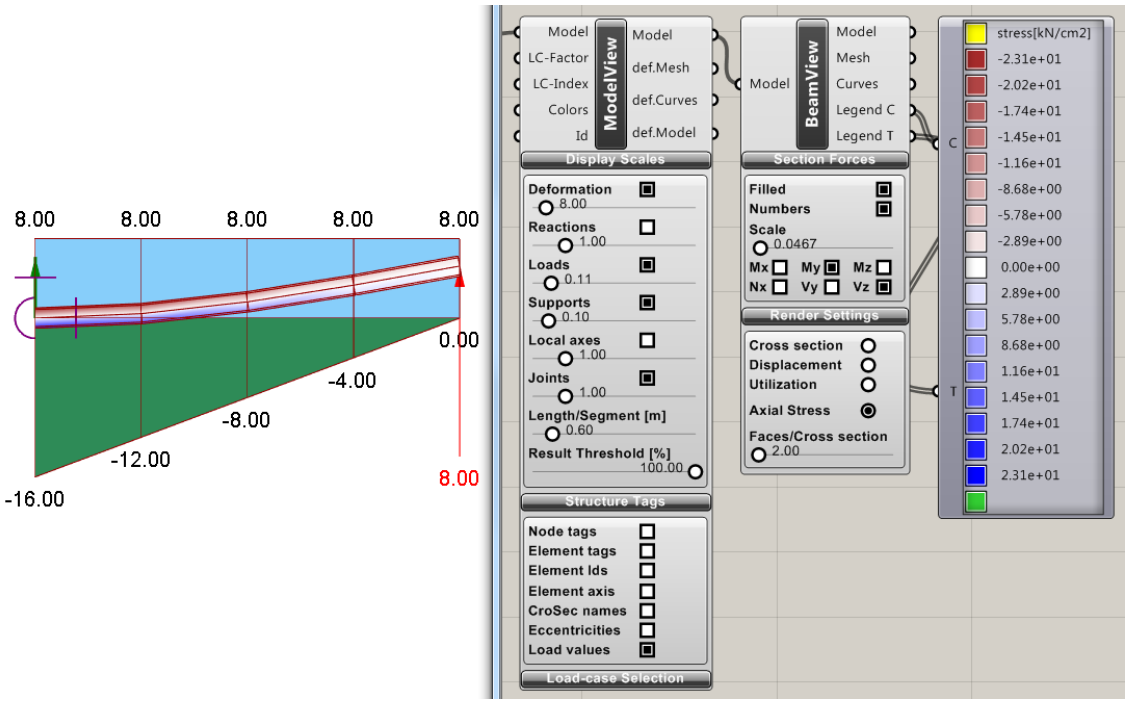


Figure 77: Moment M_y (green) about the local beam Y-Axis and shear force V_z (blue) in local Z-direction.

6.6.9. Resultant Section Forces

The "Res-S-Force"-component retrieves axial forces N, resultant bending moments M and shear forces V for all beams and load cases. See fig. 78 for the definition of N, V and M. The sequence of element results corresponds to the sequence of beams. Thus the data can be used for cross section design of radially symmetric elements.

Figure 79 shows a simply supported beam with two load cases presented in one picture. The beam consists of two elements and has a total length of eight meters. In load case zero a vertical force of magnitude $1kN$ acts vertically downwards in the middle of the beam. Load case one consists of a point-load of $3kN$ directed parallel to the undeformed beam axis. The results at the output-plugs "N" and "M" in fig. 79 are trees that hold the beams normal force in kilo Newton [kN] and resultant bending moment in kilo Newton times meter [kNm] respectively. There is only one model fed into the S-Force component thus the third index from the right is zero. The second index from the right refers to the load case: the first two lists contain results for load case zero, the last two for load case one. The last index

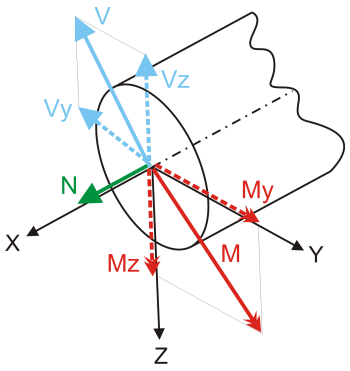


Figure 78: Normal force N, shear force V and resultant moment M at a cross section with local coordinate axes XYZ. Force and bending moment components are positive in the direction of the local coordinate axes.

corresponds to the element indices in the model.

Tensile normal forces come out positive, compressive normal forces have negative sign. The resultant moment yields always positive values as it is the length of the resultant moment vector in the plane of the cross section.

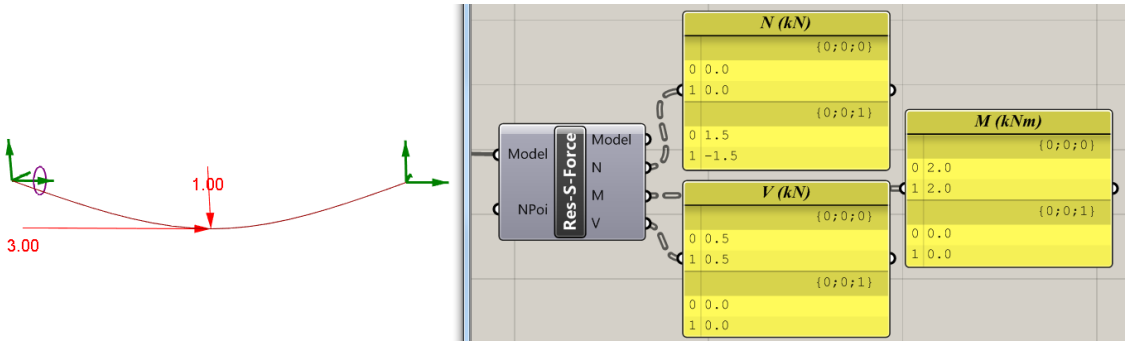


Figure 79: Simply supported beam under axial and transverse point-load: List of normal forces, shear forces and moments for all elements and all load cases.

The input-plug "NPoi" sets the number of equidistant points along the beam axis where resultant forces are calculated in order to determine the maximum values for output. In case of zero gravity and in the absence of uniform beam loads the maximum values of M and N occur at the endpoints. Otherwise these maxima may lie inside the elements. The default value of "NPoi" is three which means that values are checked at the beams end-points and in the middle.

As M is always rendered positive the maximum along an element is unambiguously given. Under gravity normal forces in a beam may change sign.

In such a case Karamba returns that N which gives the maximum absolute value.

Fig. 79 shows the results of a simply supported beam consisting of two elements under two load-cases: In load case zero both elements return zero normal force because there acts no external axial load. The maximum moment of both elements is $2[kNm]$. For a simply supported beam under a mid-point transverse load the maximum moment occurs in the middle and turns out to be $M = F \cdot L/4 = 1[kN] \cdot 8[m]/4 = 2[kNm]$.

The axial force of $3[kN]$ in load case one flows to equal parts into both axial supports. It causes tension ($1.5[kN]$) in the left element and compression ($-1.5[kN]$) in the right one.

6.6.10. Section Forces

Sometimes it is desirable to have section forces and moments represented as components in the direction of the local axes of the cross section instead of resultant values. Use the "S-Force"-component in such a case. Its output is similarly structured as that of the Res-S-Force-component described above, but its output plugs comprise the force components in local directions (see fig. 80). The input parameters "maxL" and "NRes" determine the number of results along the beam axis. "maxL" can be used to control the maximum distance between results. A negative value for "maxL" means that there is no maximum distance condition. "NRes" sets the number of results along the beams. The beams endpoints are automatically included in the output.

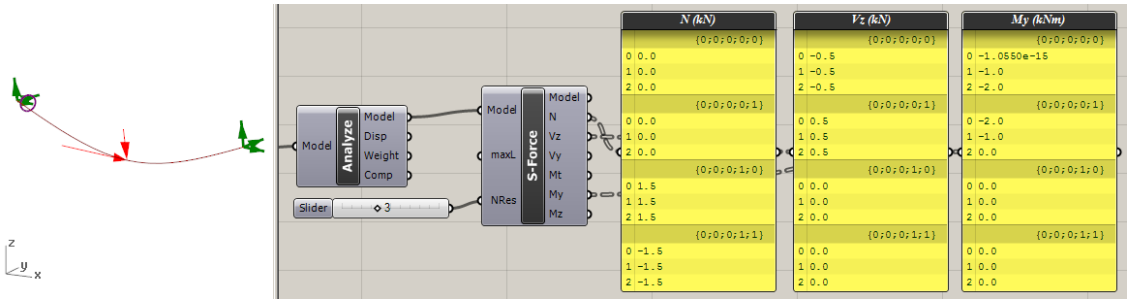


Figure 80: Simply supported beam under axial and transverse point-load: List of normal forces, shear forces and moments for all elements and all load cases along an the elements.

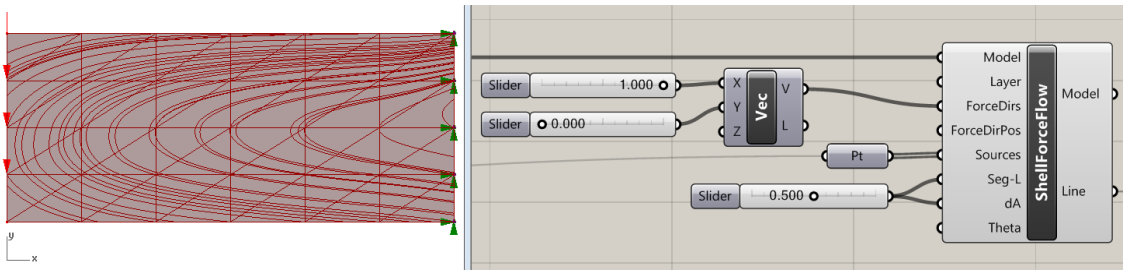


Figure 81: Cantilever consisting of triangular shell elements: Flow lines (green) of force in horizontal direction.

6.6.11. Force Flow Lines on Shells

Force flow lines or load pathes (as they are also sometimes called) illustrate the load distribution in structures [3]). There is a loose analogy between those force flow (FF) lines and streamlines in hydromechanics: The law of conservation of mass in hydromechanics is matched by the static conditions of equilibrium in a specified direction. If there are two FF-lines the resultant force between those in a predefined direction stays constant. Consider e.g. the cantilever in fig. 81 for which the force flow in horizontal direction is described by the red lines. At the supports the force flow lines run nearly horizontal at the upper and lower side where the normal stresses from the supports reach their maximum and thus dominate the resultant force. They gradually curve down to the neutral axis where the shear stresses constitute the only contribution to horizontal forces.

Aside from resulting in nice line drawings those force flow lines can be practical as well [3]:

- FF-lines form eddies in ineffective (with respect to the given force direction) parts of a structure or reverse their direction there.
- In case you want to strengthen a structure with linear elements (e.g. fibres) align them with FF-lines to get the most effective layout.

FF-lines are not the same as principal stress lines because the latter lack the property of constant force between adjacent lines.

The "Shell Force Flow Lines"-component lets you create force flow lines in arbitrary points of shells (see fig. 81). There exist seven input plugs:

- "Model": The model from which you want to create FF-lines. By default the results of all load-cases get superimposed with factor "1". Use a

"ModelView"-component to select specific load-cases or to impose load-factors other than "1".

- "Layer": In case of bending the stress state of shells and therefore the FF-lines change over the cross section height. A value of "-1" denotes the lower "1" the upper shell surface and "0" the middle layer. The default value in "0".
- "ForceDirs": Expects a vector or list of vectors that defines the direction of force. This direction gets projected on each element in order to define the local force flow directions. Elements perpendicular to the "ForceDir"-vector are skipped. Multiple such directions can be defined for different regions.
- "ForceDirPos": For each vector in "ForceDirs" a position can be defined. The force direction at an arbitrary point on the shell corresponds to the "ForceDir"-vector with the closest "ForceDirPos".
- "Source": Defines points on the shell where FF-lines shall originate. You can feed points on or near the shell into this plug. It is also possible to use lines that intersect the shell. In case of multiple intersections there will be the same number of FF-lines.
- "Seg-L": Intended length of the segments of the resulting FF-lines. Is $0.5[m]$ by default.
- "dA": This parameter sets the accuracy with which the FF-lines get determined: It is the maximum differential angle between to adjacent pieces of a FF-line. If this criteria results in pieces of length smaller than "Seg-L" then they will be joined before sent to the output-plug "Line". By default this value is set to $5[deg]$.
- "theta": Here you can define an angle between the FF-lines and those lines output at the "Line"-output plug. The angle is in $[deg]$ and defaults to zero.

The output of the "ShellFFlow"-component consists of lines arranged in a data tree. The right-most dimension contains the branches of each flow-path: In case of a e.g. a plane there are two branches that originate from the given intersection point. In case of T-like shell topologies this number can grow to three and larger.

The load-case considered is that defined in the nearest upstream "ModelView"-component.

6.6.12. Isolines on Shells

The "Isolines on Shells"-component lets you do two things: First draw contour lines on shells that connect points of equal principal stresses, utilization or resultant displacements (see fig. 82). Second query results in arbitrary points of the shell.

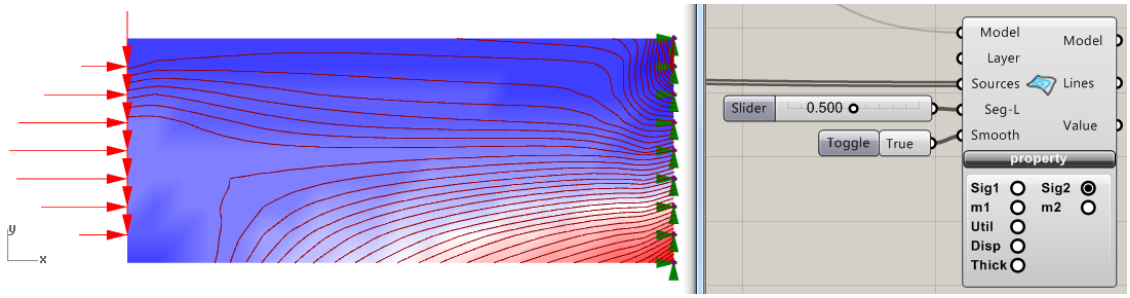


Figure 82: Lines of equal first principal stress on a cantilever.

The input-plugs "Model", "Layer", "Sources" and "Seg-L" have the same meaning as for the "Force Flow Lines on Shells"-component (see sec. 6.6.11). The load-case to examine as well as load-case factors can be set with a "ModelView"-component plugged into the definition ahead of the "Isolines on Shells"-component. By default all load-cases get superimposed using unit load-factors.

Isolines are straight lines within each shell element. This may result in slightly rugged poly-lines. Set the "Smooth"-input plug to "true" in order to flatten them out. The "Line"-output-plug will then return splines instead of lists of line-like curves. They result from using the calculated iso-points as control-points. For curved shell geometries this has the disadvantage that those splines no longer stay exactly on the shell surface. This may give you a hard time trying to intersect different groups of such lines.

In the "property"-submenu you can select the result-value to be displayed: first or second principal stress ("Sig1", "Sig2"), first or second principal bending moments ("m1", "m2"), utilization ("Util")), resultant displacement ("Disp") or shell thickness ("Thick").

The "Lines"-output data-structure corresponds to that of the "Force Flow Lines on Shells"-component. Each number in the output-plug "Value" cor-

responds to one piece of isoline from the "Lines'-output.

6.6.13. Principal Force Directions on Shells

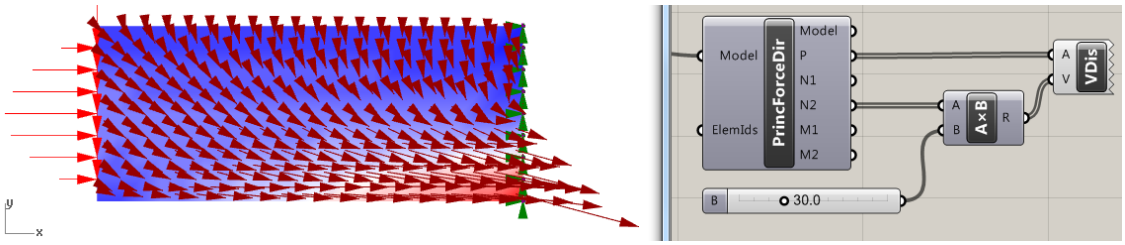


Figure 83: Cantilever analyzed as shell structure: directions of second principal normal forces at element centers.

The "Principal Force Directions on Shells"-component lets you retrieve principal normal forces and moments at element centers (see fig. 83). All shells of a model get considered by default. Use the input-plug "ElemIds" to select a subset. Results refer to the selected load-case at the nearest upstream "ModelView"-component.

The order of all result lists corresponds to the order of faces in the mesh used to generate the shell. Output-plug "P" lists the coordinates of the element centers where the normal forces and bending moments were calculated. "N1" and "N2" deliver the first and second principal normal force directions as vectors. Their length corresponds to the absolute value of the corresponding quantity in $[kN/m]$. The output plugs "M1" and "M2" return first and second principal bending moment directions.

6.6.14. Principal Moment Lines on Shells

Works like the "Principal Stress Lines on Shells" component (see section 6.6.16). Instead of principal stress lines it returns principal moment lines.

6.6.15. Principal Stress Directions on Shells

This components provides the same results as the "Princ. Stress 1-2" option of the "ShellView"-component (see fig. 84). The output-plug "P" renders the positions of the elements centroids. "V1" and "V2" return corresponding vectors for the first and second principal stresses there. Thin out results by setting "Result Threshold" of ModelView (needs to be upstream of the data-flow) to a value of less than 100% . Like for isoline and force-flow

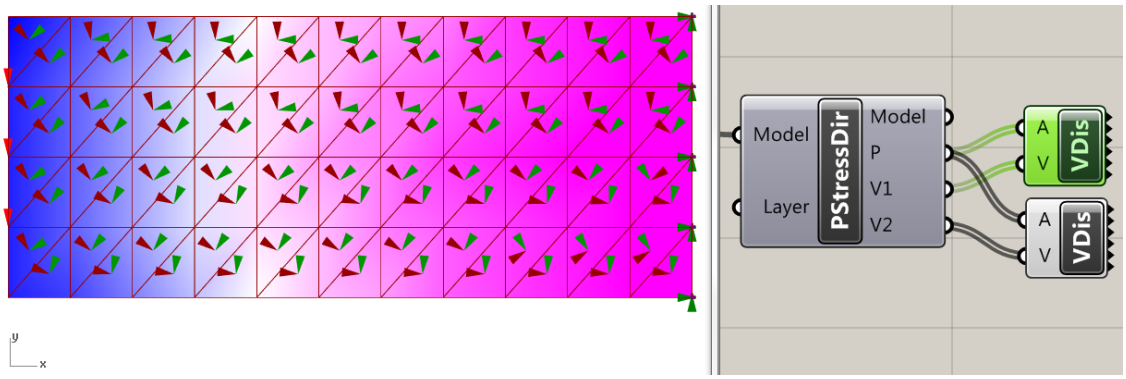


Figure 84: Triangular mesh of shell elements and principal stress directions at their centroids. Colors indicate the resultant displacement.

lines a specific load case or superimposition of load cases can be set via ModelView. By default all load-case results get added up using unit load factors.

6.6.16. Principal Stress Lines on Shells

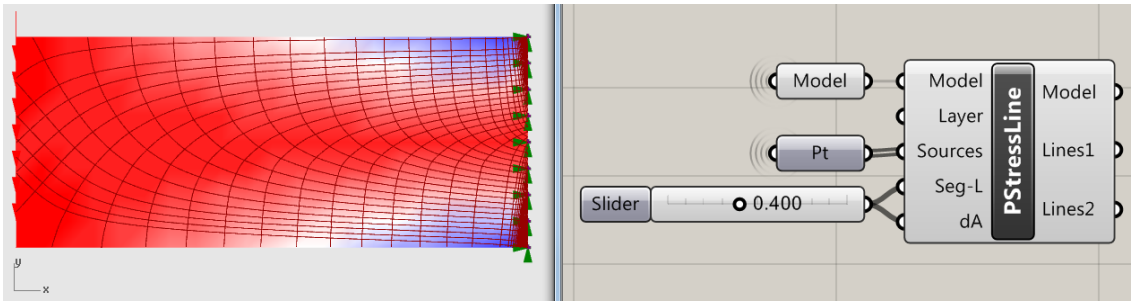


Figure 85: Principal stress lines: they are tangent to the first and second principal stress direction. The coloring reflects the level of material utilization.

Principal stress (PS) lines are tangent to the principal stress directions (see fig. 85). In the case of a cantilever they either run parallel or at right angle to the free boundaries. In the middle where normal stresses due to bending vanish first and second principal stress lines intersect at $90[deg]$.

The meaning of the input-plugs of the "Principal Stress Lines on Shells"-component correspond to that of the "Force Flow Lines on Shells"-component (see sec. 6.6.11 for details). On the output side "Lines1" and "Lines2" hold the first and second principal stress lines in data trees: the right-most dimension holds a list of lines that represent a part of a PS-line. There are

usually two parts per line that start off to either side of the starting point. In case of more complicated topologies there can be more than two parts. These parts populate the second dimension from the right.

6.6.17. Shell Forces

Sometimes it is of interest to know not only the directions (see section 6.6.13) but also the value of principal normal forces and moments on shells. In such cases the "Shell Forces"-component comes in handy. It lists the results in the same order as the "Principal Force Directions on Shells"-component. Thus the element centers returned there can be used in combination with the numeric results.

Distributed normal forces are negative in case of compression. Positive bending moments result in tension on the upper side of a shell. The upper side of a shell element is defined by a positive value of the local Z-axis. When in doubt about the orientation of your shell elements enable the preview of local element axes in the "ModelView"-component (see section 6.6.2).

6.6.18. ShellView

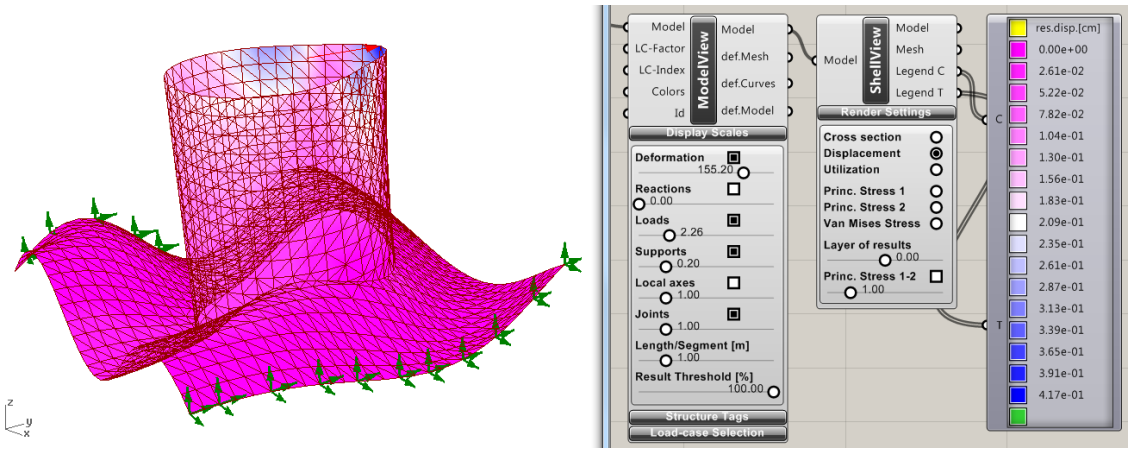


Figure 86: Resultant displacement of a shell.

The "ShellView"-component works like the "BeamView"-component (see sec. 6.6.8 and controls the display of shell results. Figure 86 shows the resultant displacement of a shell.

You can choose from these rendering options:

- "Cross section": shows the upper and lower shell surface and adds them

to the output at the "Mesh"-output plug.

- "Displacement": colors the shell according to the resultant displacement.
- "Utilization": renders the material utilization of shells. The utilization is calculated as the ratio between yield stress of the material and maximum Van Mises Stress along the shells cross section hight. The Van Mises Yield criterion is not applicable to brittle materials like concrete.
- "Princ. Stress 1": visualizes the resultant value of the first principal stress in the plane of the shell.
- "Princ. Stress 2": displays the resultant value of the second principal stress. direction would also be a correct result.
- "Van Mises Stress": renders the Van Mises Stress.
- "Layer of Results": sets the shell layer on which results are calculated. A value of $+1$ corresponds to the upper surface, -1 to the lower.
- "Princ. Stress 1-2": Enables/disables and scales the vector display of principal stresses in the element centers. Use the "Result Threshold" sliders in the "Display Scales"-menu of the "ModelView"-component to thin them out if necessary.

6.7. Export

Karamba is not meant to be a substitute for a full blown structural engineering finite element software package. Instead it aims at providing flexibility in testing different structural designs that the more traditional FE-applications lack. We therefore started to implement interfaces to those traditional civil engineering packages. At the moment Karamba supports data exchange with RStab5, RStab6, RStab7 and Robot. Interfaces to other applications will be added in near future.

6.7.1. Export Model to DStV

Communication between Karamba and RStab or Robot works via "DAStV"-file which is a "STEP"-derivative.

In order to create an exchange file place a "Export Model to DStV"-component on you definition and feed it with a model via the "Model"-plug. The "Path"-plug lets you chose a name for the exchange-file. RStab names

cross sections and materials depend on the selected language. Therefore set RStab to "English" before importing Karamba DAStV-files.

The output-plug "DAStV" returns the export-file as a list of strings which can be viewed with a "Panel"-component. Check the output of the "Info"-plug to see whether Karamba encountered problems during export.

The different versions of RStab interpret some aspects of DAStV data differently. For example distributed, projected loads will not be imported correctly in all versions of RStab. Therefore you should always compare the structural response calculated with the exported model in RStab to that obtained with Karamba.

6.8. Utilities

6.8.1. Mesh Breps

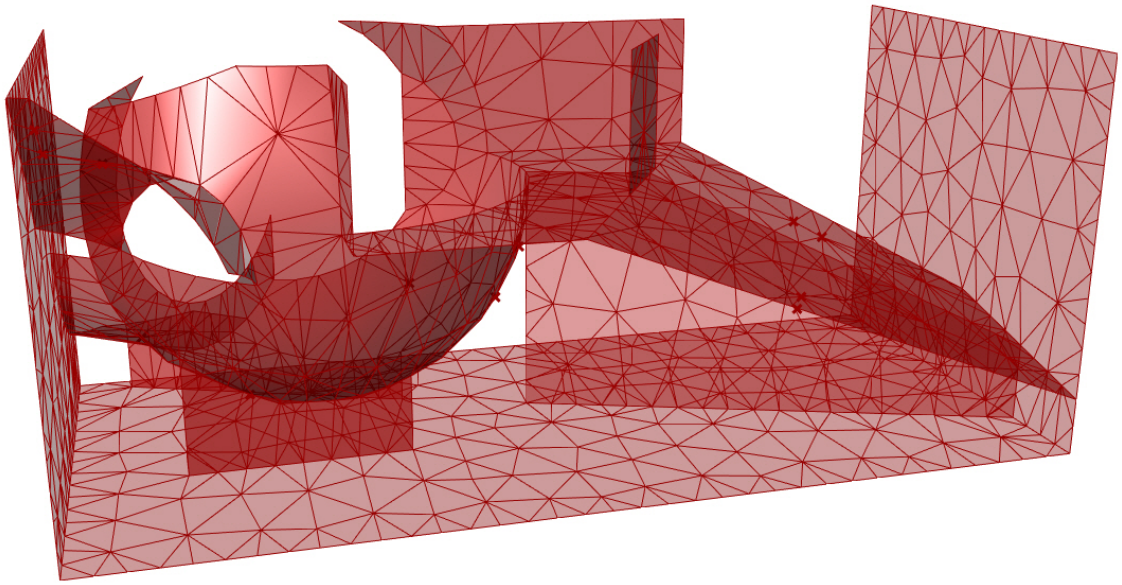


Figure 87: Unified mesh generated from Breps using the "MeshBreps"-component; created by Moritz Heimrath.

Mesh Breps[1]: In Karamba the geometry of shells is represented by meshes. Each mesh face corresponds to a constant strain finite element. The element nodes determine its connectivity to the other parts of the structure. Thus point-loads, supports, point-masses and the like can only be attached to mesh vertices. When two meshes have common boundaries they need to

[1]The "Mesh Breps" component was programmed by Moritz Heimrath.

have identical vertices there in order to be structurally connected.

The "MeshBreps"-component ensures the connectedness of meshes generated from multiple Breps. It also allows to define points on those Breps where mesh vertices shall result. Fig. 87 shows the unified mesh based on four Breps and one predefined point.

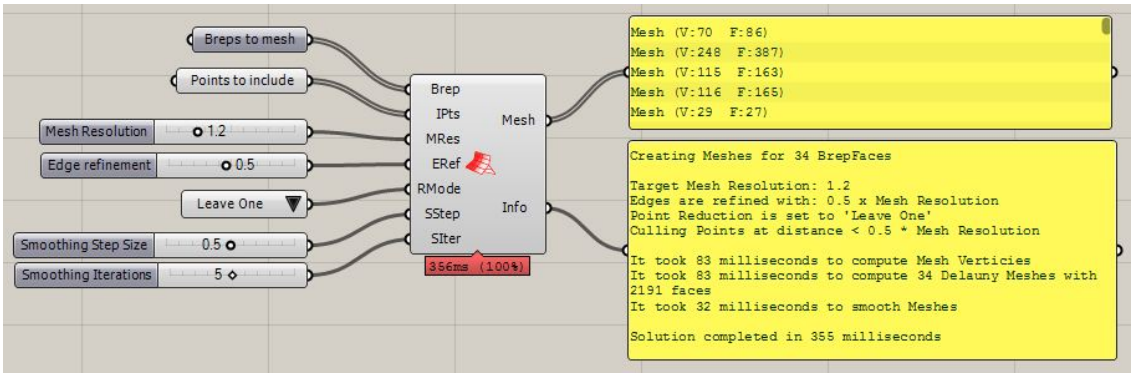


Figure 88: In- and output of the "MeshBreps"-component; created by Moritz Heimrath.

These input-plugs control mesh generation (see fig. 88):

- "Brep": List of Breps to be joined and meshed.
- "IPts": Points on Breps or at their boundaries where mesh vertices shall be generated.
- "MRes": Target size of mesh faces in meter.
- "Edge Refinement Factor": Multiplication factor for "MRes" that determines the target edge length of faces at Brep-boundaries.
- "Point Reduction": If "True" vertices in overly dense areas get culled. Such regions may result from a distorted UV-space in the underlying Brep. Points closer than half of "MRes" to their next neighbor get removed.
- "SStep" and "SIter": These two parameters let you control mesh relaxation. Triangular finite shell elements give better results when having sides of approximately equal length. The smoothing algorithm tries to improve the mesh in that direction: It moves each vertex towards the center of gravity of those vertices it connects to. During mesh relaxation vertices always remain on the Brep they belong to.

The output of the "MeshBreps"-component consists of meshes with identical vertices at common boundaries. The "Info" output-plug provides information regarding the meshing process.

6.8.2. Nearest Neighbors

Assume you have two lists of points: say "FromPts" and "ToPts". Further imagine you want to have a network that connects each point of the first set with a predefined number of its nearest neighbors in the second set (or to points in set two that lie within a given distance). In that case the "Nearest Neighbor"-component will be the right choice. It outputs lists of connection lines and a corresponding connectivity diagram. Be aware of the fact that these lists will probably contain duplicate lines. But this is no big problem (see section 6.8.4).

6.8.3. Multi-dimensional Nearest Neighbors

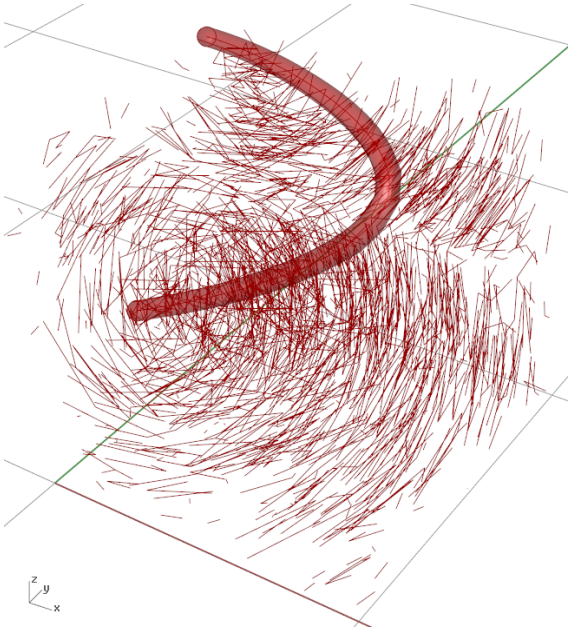


Figure 89: Random points in a unit volume connected to their nearest neighbor in a 5-D setting

A nearest neighbor search can be generalized to any number of dimensions. Use the "Multi-dimensional Nearest Neighbors"-component in case of more than three. Fig. 89 shows an example with five dimensions: Dimensions one to three represent space. Dimension four is the shortest distance between

each point and the thick red guide line. The curve parameter of the guide line at the point where it meets the line of shortest distance acts as fifth dimension. Each of the randomly generated points is connected with its nearest neighbor. One can see from fig. 89 that the resulting line segments align to the guide curve – in some way.

There are three input-plugs on the component:

- "FromPts": expects a list of lists, where the rightmost list comprises n values which are the coordinates of the point. The 'FromPts"-input specifies points where nearest neighbor connections can start.
- "ToPts": expects the same sort of input as "FromPts". It contains the points where nearest neighbor connections can end.
- "NN": number of nearest neighbor connections to be generated for each point in "FromPts"

The output "C" of the "Multi-dimensional Nearest Neighbors"-component is a connectivity diagram.

6.8.4. Remove Duplicate Lines

When you have a list of lines that you suspect of containing duplicate lines then send it through this component and out comes a purified list of ones of a kind. The input-plug "LDist" determines the limit distance for nodes to be considered as identical. Lines of length less than "LDist" will be discarded. Lines that overlap only partly can not be detected.

6.8.5. Remove Duplicate Points

Does essentially the same as the component described above – only with points instead of lines.

6.8.6. Get Cells from Lines

This component takes a connectivity diagram and corresponding nodes and determines all closed cells. Input plug "Pl" can be used to supply planes that are approximately parallel to the cells. The result is a connectivity diagram of the closed cells.

6.8.7. Line-Line Intersection

This component takes a list of lines as input and mutually tests them for intersection. Output-plug "IP" delivers a list of intersection points. "LSS" returns all the line-segments including those which result from mutual intersection. The input-plug "LTol" sets the tolerance length in meter for finding intersections between lines that do not actually intersect. "LTol" is also the minimum length of segments at the ends of lines (e.g. at T-intersections).

6.8.8. Element Felting

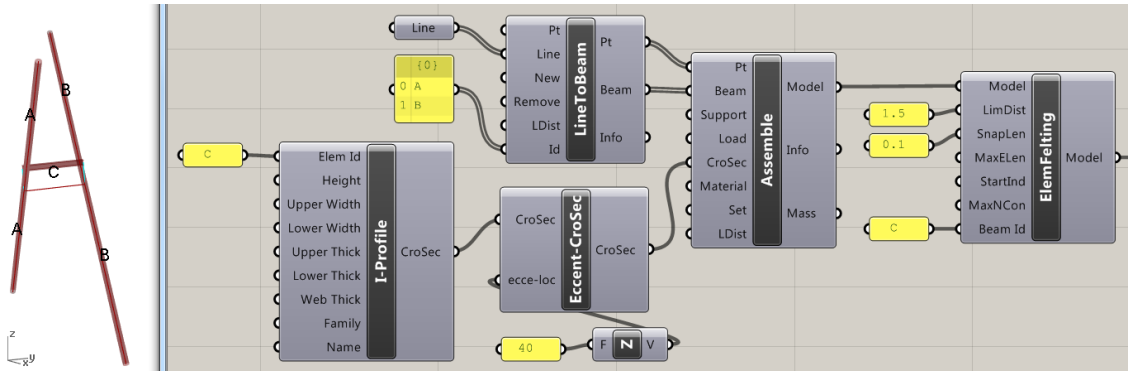


Figure 90: The elements "A" and "B" of the original model are connected resulting in the additional element "C".

Sometimes one has several (potentially) structural elements neatly positioned in space but no connections between them. The "Element Felting"-component helps out in such situations by generating connections between neighboring elements (see fig. 90). The components behavior can be controlled with these input-plugs:

- "Model": Karamba model to be dealt with
- "LimDist" $[m]$: The "Element Felting"-component calculates the shortest distance between each pair of elements in the model. If this distance is less than "LimDist" a connection will be generated.
- "SnapLen" $[m]$: In case that a connection is to be generated the participating elements need to be divided and a connection element introduced. If any of the thus arising elements has a length of less than "SnapLen" then the element will be removed and its endpoints snap to the older point of the two.

- "MaxELen" $[m]$: you can set here a length limit for elements that shall take part in the felting-process. All element longer than the value of "MaxELen" will be ignored.
- "StartInd": Lets you limit the felting process to elements with an index larger than or equal "StartInd".
- "MaxNCon": This sets the maximum number of new connections to be generated. If this value is reached then felting simply stops.
- "Beam Id": The beam identifier provided her will be attributed to the connections generated by the component. Cross sections, materials and eccentricities previously defined for this beam identifier apply to these. In case no identifier is given neighboring elements snap to the point in the middle of their shortest connection line.

The felting algorithm proceeds from the first element to the last, always testing against all currently existing elements. Therefore newly generated connection elements may be the source of further connections.

The algorithm operates directly on the Karamba-model and is therefore quite fast.

6.8.9. Mapper

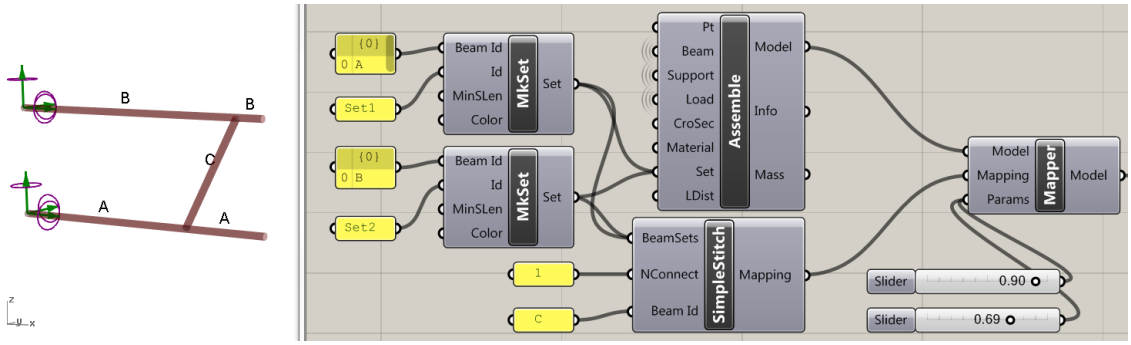


Figure 91: The "Mapper"-component applies mappings to a given model. In this case there is one mapping that connects two beam-sets with elements whose position is controlled by the parameters given to the mapper.

A "Mapper" is a component that takes a Karamba-model and modifies it according to some generic rules defined by mappings, based on parameters supplied by the user. It acts directly on the Karamba-model, so the process of transferring Grasshopper-geometry to a Karamba-model is dispensed with.

The resulting gain of speed can be important when running optimization tasks with e.g. Galapagos.

Fig. 91 shows a definition where a mapper applies a mapping called "Simple Stitch" on a given model which originally consists of two elements: "A" and "B". The input-plug "Params" receives two parameters. In the context of the "Simple Stitch"-mapping these parameters give the relative position on the two beam-sets "A" and "B" where a connection "C" shall be introduced. So a mapping encapsulates an operation and the mapper activates it [10].

Currently Karamba offers mappings which mainly deal with connecting existing beam-sets by variants of an operation termed "stitching". The notion comes from the analogy to joining together pieces of cloth. These mappings will be explained further below. They are rooted in the research project "Algorithmic Generation of Complex Space Frames" which was conducted at the University of Applied Arts Vienna.

6.8.10. Interpolate Shape

The "Interpolate Shape"-component allows one to span a design-space with basic means: one Karamba-model acts as origin of that space; an arbitrary number of other models define a coordinate axis each. Linear interpolation takes place between the model at the origin and those defining the axes. A parameter value of 0.0 corresponds to the origin, 1.0 to the model defining the corresponding axis.

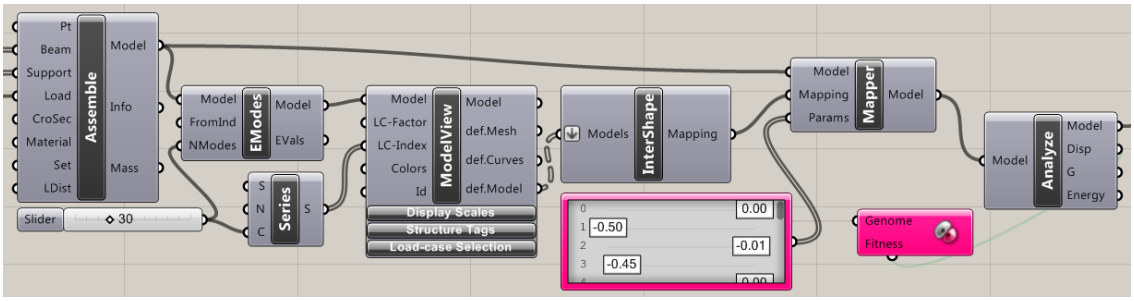


Figure 92: Definition for optimizing the shape of a simply supported beam under mid-span single load.

Fig. 92 shows a definition where the first 30 eigen-forms (the thin red lines in 93) of a simply supported beam serve as the shape-dimensions of the design space.

[10] In other words: a mapping is a functor.

Galapagos is used to determine the position in that design-space which results in minimum deflection under a single load at mid-span. It is clear (think of a hanging model) that the optimum shape has a sharp kink under the load and is otherwise straight.

Fig. 93 shows the result of the optimization run which resembles the ideal shape to a large degree. A sharper bend underneath the load could be achieved by including more shape-dimensions in the design space.

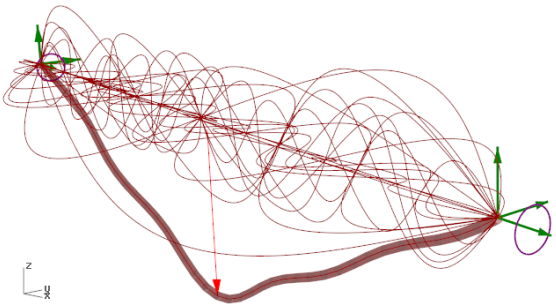


Figure 93: Result of shape optimization (thick red line) for a simply supported beam under mid-span single load using the first 30 eigen-forms – the thin red lines – as axes of the design space.

6.8.11. Proximity Stitch

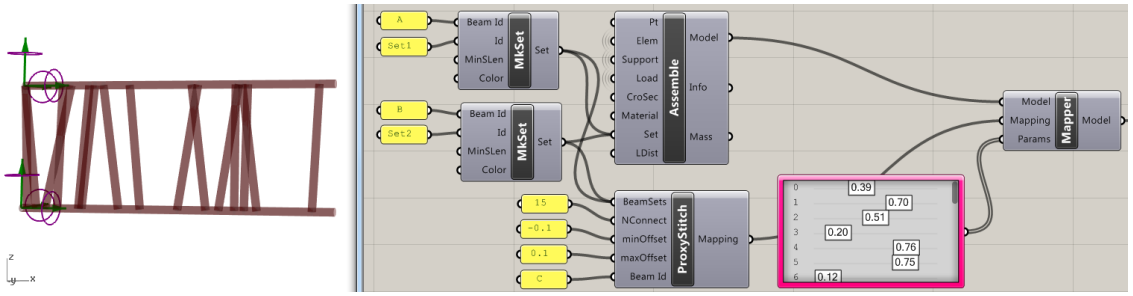


Figure 94: "Proximity Stitch"-mapping with the same set-up as in fig. 91 but fifteen random connections instead of two.

The "Proximity Stitch" is a tamed "Simple Stitch" (see sec. 6.8.12): In case of n beam-sets a tuple of n parameters describes one connection. All parameters are in the range $[0, 1]$. The first value p_1 sets the relative location l_1 on the first beam-set. All following parameters p_n relate to the restricted interval $[l_{n-1} - minOffset, l_{n-1} + maxOffset]$. Here "minOffset" and "maxOffset" can be defined by the user. The narrower the interval they define, the

more regular the structure.

6.8.12. Simple Stitch

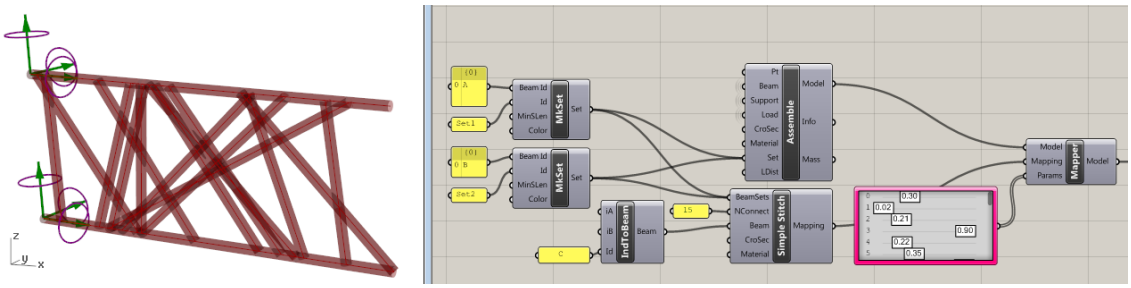


Figure 95: "Simple Stitch"-mapping with the same set-up as in fig. 91 but fifteen random connections instead of two.

A "Simple Stitch"-mapping connects two or more beam-sets with truss or beam-elements. The input-plug "BeamSets" expects a list of beam-sets which get connected in the order as they are listed. Double entries of sets are no problem. Via "NConnect" one sets the number of connections. There needs to be one parameter per beam-set and connection for specifying the mapping. The numerical range of parameters should be zero to one: zero is the starting position of the beam-set, one its end. In case you fail to provide the mapper with a sufficient number of parameters it will turn red. Read its error-message in order to see how many parameters are needed. The input plug "Beam Id" can be used to define the name of the connection elements. Fig. 95 shows a structure with 15 connections resulting from 30 randomly selected parameters.

This "simple"-variant of stitches is also the most versatile one: it gives you great freedom in generating connection patterns by defining the way how a set of parameters is mapped to the set of values that are fed into the "Simple Stitch". Sections 6.8.11 and 6.8.13 deal with variants of the "Simple Stitch" which are limiting the scope of possible patters. This approach leads to faster convergence in case of optimization with e.g. Galapagos and spares you programming effort but lacks the full flexibility of the "Simple Stitch".

6.8.13. Stacked Stitch

A "Stacked Stich"-component works along the same lines as a "Simple Stitch". The difference is, that it maps the given parameters to a geometry in such

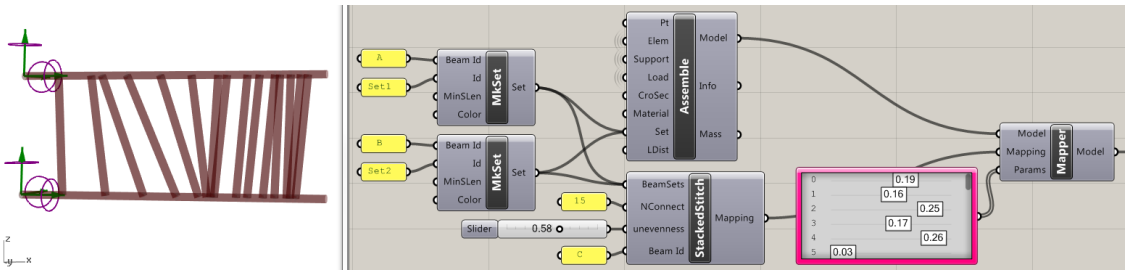


Figure 96: "Stacked Stitch"-mapping with the same set-up as in fig. 91 but fifteen random connections instead of two.

a way, that no two connection elements cross each other (see fig. 96). The input-plug "unevenness" can be used to fine-tune the average irregularity of the result. Zero means totally even: the connection elements are placed at equal distance along the beam-sets (in terms of the corresponding beam-set parameter). The larger the value in "unevenness" the more irregular the layout of connection elements.

6.8.14. User Iso-Lines and Stream-Lines

The components "User Iso-Lines" and "User Stream-Lines" let you draw iso-lines and stream-lines on arbitrary meshes by defining values and vectors at their vertices.

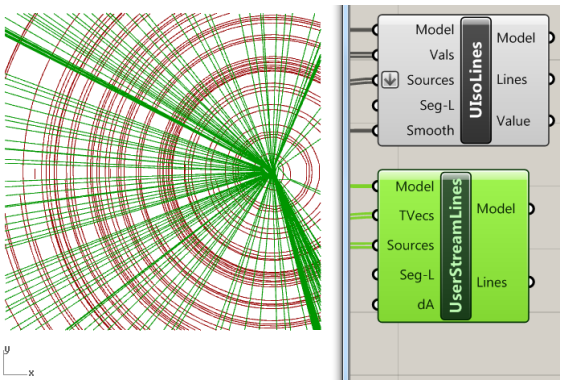


Figure 97: User defined Iso-lines (red) and stream-lines (green) on a rectangular shell patch.

The definition "Shell_UserDefinedIsoAndStreamLines.gh" in "..Libraries/-Karamba/Examples/TestExamples/" uses a function which when given a point returns the distance from and direction to a predefined source point (see fig. 97).

Both components "User Iso-Lines" and "User Stream Lines" work similar to the "IsoLines"- (see section 6.6.12) and "Principal Stress Lines on Shells"-components (see section 6.6.16) respectively. The only difference lies in the fact that for each node of the model (which includes also nodes that only connect to beams) a value (input-plug "Vals") or vector (input-plug "TVecs") tangent to the flow needs to be supplied.

7. Trouble shooting

Do not panic in case some Karamba-components turn red upon feeding them with your model. Read the error message. It usually contains information that helps you further. The very efficient way of tracking errors is to divide and conquer:

1. Split the model in half.
2. Check both parts.
3. Scrutinize the part that does not work.
4. See whether you can find an error in it.
5. If not take that part as your new model and proceed to point 1.

7.1. Karamba does not work for unknown reason

This is the recommended procedure:

1. If a component turns red read its runtime-error-message.
2. In case that more than one item is plugged into an input, check the incoming data via a panel component.
3. Sometimes flattening the input data helps: The dimension of input-lists must be consistent. For diagnosis plug them into a Panel-component which will show the dimensionality of the data. Another method is to enable "Draw Fancy Wires" in the View menu: Differently outlined connection lines signify different dimensionality of the data that flows through them.
4. If no results show, check whether preview is enabled on the ModelView, BeamView or ShellView-component.

5. If the Analyze-component reports a kinematic structure do the following:
    - Check the supports for forgotten support conditions.
    - Start to fix all degrees of freedom on your supports until the Analyze-component reacts.
    - Introduce additional supports.
    - Plug the model into the EigenModes-component. The first eigen-modes will be the rigid body modes you forgot to fix. Save your model before doing that: Large models can take a long time to calculate.
    - If the first few eigen-modes seemingly show an un-deflected structure there might be beams in the system that rotate about their longitudinal axis. Enable "Local Axes" in the ModelView-component and move the slider for scaling Deformation in order to check this.
    - Turn trusses into beams by activating their bending-stiffness. Be aware of the fact that a node has to be fixed by at least three trusses that do not lie in one plane.
    - Remember that trusses have no torsional or bending stiffness and thus can not serve to fix the corresponding rotations on a beam that attaches to the same node.
    - Check whether an element has zero area, height or Young's modulus.

7.2. "fem.karambaPINVOKE"-exception

On some computers the analysis component of Karamba refuses to work and throws a "fem.karambaPINVOKE" exception. This may be due to left-overs from previous Karamba installations which were not removed properly during the installation procedure. In such a case precede as follows:

- Uninstall Karamba completely via settings/Software/...
- Make sure that everything was removed:
    - Remove "karamba.dll" and "libiomp5md.dll" from the windows folder if they sill exist.
    - Search your hard-disk for "karamba.dll"- and "karamba.gha"-files and remove all occurrences by hand.

- reinstall Karamba.

If this does not help do the following:

- Check whether the "karamba.dll"'-file in the "Windows"-folder is blocked: right-click on the file and select "Properties" then security.
- Make sure that you installed a Karamba version with the correct bitness: Karamba (64bit) can be used together with "Rhinoceros 5 (64bit)"; Karamba (32bit) with "Rhinoceros 5". Be aware of the fact that both versions of Rhino get installed.

This is plan b if the above does not help:

- Start Grasshopper
- Type "GrasshopperDeveloperSettings" in the Rhino Window and hit "ENTER"
- Toggle the status of the "Memoryload *.GHA assemblies using COFF byte arrays" option
- Restart Rhino

Plan c is to post a help request to the Karamba group at `http://www.grasshopper3d.com/group/karamba`.

7.3. The StackedStitch-components renders structures with overlapping diagonals

Beam-sets have an orientation. You probably use beam-sets with opposing directions.

7.4. Karamba does not work after reinstalling Grasshoper

Upon installing Grasshopper some files of the Karamba package may have been erased. Try to reinstall Karamba.

7.5. Karamba does not appear nor any of its components seem to be installed

In case of multiple versions of Grasshopper on your machine make sure that you installed Karamba to that Grasshopper version which is used by Rhino. In

Rhino select "Tools/Options.../Plug-ins" from the menu, then "Grasshopper" from the list of plug-ins. Press the "Properties..."-button to check the path to Grasshopper.

Multiple Grasshopper versions can result if there are more than one users on a machine and each of them installs his or her own Grasshopper.

7.6. Karamba seems to get stuck while calculating a model

Depending on your computer (CPU, size of internal memory) Karamba can handle models in the order of 10000 elements efficiently. If overlong computation times occur check the number of models you actually calculate. Having the path structures of the input-data wrong may lead to multiple models. In such cases flattening or simplifying the input-data helps.

7.7. Predefined displacements take no effect

Check whether you disabled the correct degrees of freedom in the "Conditions" section of the PreDisp-component.

7.8. The ModelView-component consistently displays all load cases simultaneously

If the ModelView-component does not seem to react to selections done with the drop-down-list for load cases, check the value in the "LC-Index"-input plug. Remember that its value is added to the load case index selected on the drop-down-list. If the sum is negative all load cases will be displayed.

7.9. The View-components do not show rendered meshes (stress, strain,...), supports, etc.

Check whether "Shaded Preview" is enabled in Grasshoppers Solution menu.

7.10. The ModelView-component does not display any tags

Check whether your Rhino background color is black. Some types of tags are printed in black and do not show on a black canvas.

7.11. Circular cross sections show up as flat stripes when rendered

Set the "Faces/Cross section" slider of the "ModelView"-component to a value larger than two such that the displayed result sufficiently corresponds to your idea of roundness.

7.12. Icons in "Karamba"-toolbar do not show up

Sometimes it happens that Karambas component panels do not display any component icons. Select menu item "View/Show all components" in order to make them show up.

7.13. Error messages upon loading definitions saved with outdated Karamba versions

When loading definitions based on outdated Karamba version a pop-up window will inform you that "IO generated x messages,...". Normally this can be ignored. It may happen however that very old Karamba components do not load. In this case put their current versions in place. Deprecated components issue a warning-message (turn orange). In case you enable "Draw Icons" in Grasshoppers "View" menu, deprecated components are marked by a skull.

7.14. Component in old definition reports a run-time error

On some components the order of input-plugs changed over time (e.g. the Assemble-component). They will turn red when loaded and the runtime error message will state that one object can not be cast to some other object. In this case replace the old component with a new one and reattach the input-plugs accordingly.

7.15. The "Optimize Cross Section"-component does not work

Make sure that the beams you intend to optimize belong to the same family as those you want them to be selected from.

7.16. The "Optimize Cross Section"-component returns wrong results

Increase the value at the "iter"-input-plug. The cross section optimization algorithm is an iterative procedure. In case you stop too early – having limited the maximum number of iterations to a small value – the algorithm has no chance to converge and thus returns seemingly wrong results. Always check the "Info"-output of the component for information on the solution procedure.

7.17. Other problems

In case you encounter any further problems please do not hesitate to contact us at `info@karamba3d.com` or via the Karamba group at `http://www.grasshopper3d.com/group/karamba`.

A. Background information

A.1. Basic Properties of Materials

A.1.1. Material Stiffness

The stiffness i.e. resistance of a material against deformation is characterized by its Young's Modulus or modulus of elasticity E. The higher its value the stiffer the material. Table 2 lists E-values for some popular building materials.

type of material	$E[kN/cm^2]$
steel	21000
aluminum	7000
reinforced concrete	3000
glass fiber	7000
wood (spruce)	1000

Table 2: Young's Modulus of materials

For composite materials – like in the case of rods made from glass fiber and epoxy – it is necessary to defer a mean value for E using material tests. Karamba expects the input for E to be in kilo Newton per square centimeter $[kN/cm^2]$.

If one stretches a piece of material it not only gets longer but also thinner: it contracts laterally. In case of steel for example lateral strain amounts to 30% of the longitudinal strain. In case of beams with a large ratio of cross section height to span this effect influences the displacement response. In common beam structures however this effect is of minor importance. The shear modulus "G" describes material behavior in this respect.

A.1.2. Specific Weight

The value of "gamma" is expected to be in kilo Newton per cubic meter $[kN/m^3]$. This is a force per unit of volume. Due to Earths gravitational acceleration ($a = g = 9.81[kg \cdot m/s^2]$) and according to Newtons law ($f = m \cdot a$) a mass m of one kilogram acts downwards with a force of $f = 9.81N$. For calculating deflections of structures the assumption of $f = 10N$ is accurate enough. Table 3 gives specific weights of a number of typical building materials. The weight of materials only takes effect if gravity is added to a load

case (see section 6.4.6).

A.1.3. Theoretical Background of Stiffness, Stress and Strain

As mentioned in section 6.5.1 strain is the quotient between the increase of length of a piece of material when loaded and its initial length. Usually one uses the Greek letter ε for strains. Strain induces stress in a material. Stress is force per unit of area. From the stress in an beam cross-section one can calculate the normal force that it withstands by adding up (integrating) the product of area and stress in each point of the cross-section. Stress is normally symbolized by the Greek letter σ. Linear elastic materials show a linear dependence between stress and strain. The relation is called Hooke's Law and looks like this:

$$\sigma = E \cdot \varepsilon$$

E stands for Young's Modulus which depends on the material and depicts its stiffness. Hooke's law expresses the fact that the more you deform something the more force you have to apply.

A.2. Additional Information on Loads

Karamba expects all force-definitions to be in kilo Newton (kN). On earth the mass of $100kg$ corresponds to a weight force of roughly $1kN$. The exact number would be $0.981kN$ but $1kN$ is normally accurate enough. Table 3 contains the specific weight of some everyday materials. Rules of thumb numbers for loads can be found in table 4. Do not take these values too literally. For example snow loads vary strongly depending on the geographical situation.

Loads acting along lines or on a specified area can be approximated by point-loads. All you need to do is estimate the area or length of influence for each node and multiply it with the given load value. The Mesh-Load-component (see section 6.4.2) automates this task for surface loads.

A.3. Tips for Designing Statically Feasible Structures

Karamba can be used to analyze the response of structures of any scale. When using the "Analysis"-component for assessing the structural behavior be aware of two preconditions: First deflections are small as compared to the

type of material	$[kN/m^3]$
reinforced concrete	25.0
glass	25.0
steel	78.5
aluminum	27.0
fir wood	3.2
snow loose	1.2
snow wet	9.0
water	10.0

Table 3: Specific weights of some building materials

loads	
type	$[kN/m^2]$
life load in dwellings	3.0
life load in offices	4.0
snow on horizontal plane	1.0
cars on parking lot (no trucks)	2.5
trucks on bridge	16.7

Table 4: Loads for typical scenarios

size of the structure. Second materials do behave in a linear elastic manner – i.e a certain increase of deformation is always coupled to the same increase of load. Real materials behave differently: they weaken at some point and break eventually. If you want to calculate structures with large deflections you have to increase the load in several steps and update the deflected geometry. This can be done with the "Large Deformation Analysis"-component (see section 6.5.2)

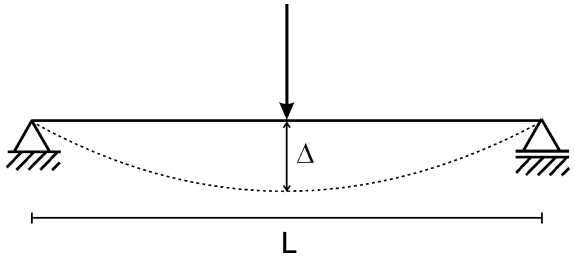


Figure 98: Simply supported beam.

For typical engineering structures the assumptions mentioned above suffice

for an initial design. In order to get meaningful cross section dimensions limit the maximum deflection of the structure.

Figure 98 shows a simply supported beam of length L with maximum deflection Δ under a single force at mid-span. The maximum deflection of a building should be such that people using it do not start to feel uneasy. As a rough rule of thumb try to limit it to $\Delta \leq L/300$. If your structure is more like a cantilever $\Delta \leq L/150$ will do. This can always be achieved by increasing the size of the cross-section. If deflection is dominated by bending (like in figure 98) it is much more efficient to increase the height of the cross-section than its area (see section 6.1.11). Make sure to include all significant loads (dead weight, live load, wind...) when checking the allowable maximum deflection. For a first design however it will be sufficient to take a multiple of the dead-weight (e.g. with a factor of 1.5). This can be done in Karamba by giving the vector of gravity a length of 1.5.

In case of structures dominated by bending, collapse is preceded by large deflections (see for example the video of the collapse of the Tacoma-Narrows bridge at `http://www.youtube.com/watch?v=3mclp9QmCGs` which also gives an impression of what normal shapes are). So limiting deflection automatically leads to a safe design. If however compressive forces initiate failure, collapse may occur without prior warning. The phenomenon is called buckling. In Karamba it makes no difference whether an axially loaded beam resists compressive or tensile loads: it either gets longer or shorter and the absolute value of its change of length is the same. In real structures the more slender a beam the less compressive force it takes to buckle it. An extreme example would be a rope. As a rule of thumb limit the slenderness – which is approximately the ratio of free span to diameter – of compressed elements to $1/100$.

A.4. Hints on Reducing Computation Time

Karamba spends most of its time solving for the deflections of a model. The time needed depends on the number of degrees of freedom n of the statical system and how many connections exist between the nodes. In the theoretical case that each node is connected to all others computation-time grows with n^3. If each node is connected to n_{neigh} others and the overall structure has a main axis along which it is oriented (i.e. there are no connections between distant nodes) then computational effort increases approximately with $0.5 \cdot n \cdot n_{neigh}^2$. Karamba makes use of multiple processors

so having more than one saves time. Using trusses instead of beams more than halves computation time. When doing optimization with Galapagos the continuous display updates slow down things considerably. For better performance disable the preview of all components or minimize the Rhino window.

A.5. Natural Vibrations and Eigen Modes

The Eigen-modes of a structure describe the shapes to which it can be deformed most easily in ascending order. The first mode is the one which can be achieved most easily. The higher the mode number the more force has to be applied. Eigen-modes find application in two areas of the engineering discipline: First in structural dynamics, second in stability analysis which involves the determination of buckling loads [9].

A.5.1. Eigen-modes in Structural Dynamics

If you hit a drum on its center the sound you hear originates from the drum-skin that vibrates predominantly according to its first normal-mode. When things vibrate then the vibration pattern and frequency depend on their stiffness and support conditions, the distribution of mass and where you hit them. Imagine a drum with a small weight placed on it: the weight will change the sound of the drum. The same is true when you hit the drum near its boundary instead of in the center: You will excite the drums eigen-modes differently. In this manual eigen-modes of a statical system that

[9] An Eigen-mode $\vec{x}$ is the solution to the matrix-equation $\underset{\sim}{C} \cdot \vec{x} = \lambda \cdot \vec{x}$ which is called the special eigen-value problem. Where $\underset{\sim}{C}$ is a matrix, $\vec{x}$ a vector and λ a scalar (that is a number) called eigen-value. The whole thing does not necessarily involve statical structures. Eigen-modes and eigen-values are intrinsic properties of a matrix. When applied to structures then $\underset{\sim}{C}$ stands for the stiffness-matrix whose number of rows and columns corresponds to the number of degrees of freedom of the statical system. $\vec{x}$ is an eigen-mode as can be computed with Karamba.

Vibration modes $\vec{x}$ of structures result from the solution of the generalized Eigenvalue problem. This has the form $\underset{\sim}{C} \cdot \vec{x} = \omega^2 \cdot \underset{\sim}{M} \cdot \vec{x}$. In a structural context $\underset{\sim}{M}$ is the mass-matrix which represents the effect of inertia. The scalar ω can be used to compute the eigen-frequency f of the dynamic system from the equation $f = \omega/2\pi$. In the context of structural dynamics eigen-modes are also called normal-modes or vibration-modes. If the eigenfrequency has a low value this means that the vibration takes a long time for one cycle to complete. In the limit when $f = 0$ this means that the static system leaves its initial state of equilibrium and never returns. In other words the system gets unstable. As the value of f goes towards zero so does the speed of vibration. This means that inertia forces do not play an important role.

include the effect of mass (inertia) will be referred to as "natural vibration modes". See sec. 6.5.4 for how to calculate them.

A.5.2. Eigen-modes in Stability Analysis

An unstable structure can be thought of as a statical system that vibrates very slowly, leaves its initial state of equilibrium and never comes back. This is why inertia forces do not play a role in the calculation of buckling-shapes. Otherwise the phenomena of vibration and buckling are very similar: The loads placed on a structure together with the eigen-modes determine the actual way a structure buckles.

The Euler-cases describe the buckling modes of beams subject to different support conditions. The more degrees of freedoms fixed at the boundary the higher the load at which a beam buckles.

Currently the stiffness matrix calculated in Karamba does not include the stabilizing or destabilizing effect of axial forces. Therefore the resulting eigenvalues do not correspond to the load-factors for buckling.

A.6. Approach Used for Cross Section Optimization

Karamba does cross section design by going through the list of cross sections in a group of cross sections called a family. It starts at the first entry and proceeds to the next until a cross section is found that is sufficient for the given cross section forces.

The core routine of the cross section optimization is the function "isSufficient(...)" which is listed below. It returns "True" if the cross section resistance surpasses the acting cross section forces. This is the original C++ listing of the variant for elastic cross section design which makes use of the elastic resisting moments W_y and W_z:

```
1  bool ElemSLDimElastic :: isSufficient(ElementStraightLine& beam,
2      const Beam3DProperties* props, int lc) const {
3
4      // assumptions:
5      //      no lateral, torsional buckling
6      //      interaction coefficients kyy, kyz, kzz = 1.0
7      //---
8
9      real lk    = bucklingLength(beam.id());
10
11     if (lk <= 0) return true;
12
13     real fy    = props->fy;
14     real E     = props->E;
15
16     real h     = M_PI * M_PI * E / (lk * lk);
17     real Ncry  = h * props->Iyy;
18     real Ncrz  = h * props->Izz;
```

```
19          h     = sqrt(props->A * fy);
20      real lam_y = h / sqrt(Ncry);
21      real lam_z = h / sqrt(Ncrz);
22      real phi_y = 0.5*(1 + props->alpha_y * (lam_y - 0.2) + lam_y * lam_y);
23      real phi_z = 0.5*(1 + props->alpha_z * (lam_z - 0.2) + lam_z * lam_z);
24      real phi   = phi_y > phi_z ? phi_y : phi_z;
25      real lam   = phi_y > phi_z ? lam_y : lam_z;
26      real chi   = 1/(phi + sqrt(phi*phi - lam * lam));
27           chi   = chi > 1 ? 1 : chi;
28
29      real Nrd   = props->A * fy;
30      real Nbrd  = chi * Nrd;
31      real Myrd  = Wy(props) * fy;
32      real Mzrd  = Wz(props) * fy;
33      real Mtrd  = props->Wt * fy / sqrt(3.0);
34      real Vyrd  = props->Ay * fy / sqrt(3.0);
35      real Vzrd  = props->Az * fy / sqrt(3.0);
36
37      // cross section with zero resistance is insufficient
38      if (Nrd == 0 || Nbrd == 0 || Myrd == 0 || Mzrd == 0 ||
39          Mtrd == 0 || Vyrd == 0 || Vzrd == 0) {
40          _insufficiency += 1;
41          return false;
42      }
43
44      // for nodes i and k
45      for (int node_ind=0; node_ind<2; node_ind++) {
46          VecReal force = beam.localForce(lc, node_ind);
47          real N  = force[Node::x_t];
48          real Vy = force[Node::y_t];
49          real Vz = force[Node::z_t];
50          real Mt = force[Node::x_r];
51          real My = force[Node::y_r];
52          real Mz = force[Node::z_r];
53          if (node_ind!=0) N = -N; // tension is > 0
54
55          // for axial stress
56          real fb;
57          if (N>0)
58              fb  = N/Nrd  + abs(My/Myrd) + abs(Mz/Mzrd) + abs(Mt/Mtrd);
59          else
60              fb  =-N/Nbrd + abs(My/Myrd) + abs(Mz/Mzrd) + abs(Mt/Mtrd);
61
62          // for shear
63          real fs = abs(Vy/Vyrd) + abs(Vz/Vzrd) + abs(Mt/Mtrd);
64
65          // take the larger of the two
66          real f = fb > fs ? fb : fs;
67
68          // update utilization level
69          _utilization = _utilization > f ? _utilization : f;
70
71          if (f>1) {
72              _insufficiency += f;
73              return false;
74          }
75      }
```

In line 9 the buckling length l_k is determined. The function "bucklingLength" starts with the end-points of the given beam and procedes to its neighbors until nodes with more than two beams attached are detected. The distance between these nodes is taken as the buckling length.

In lines 13 and 14 the materials yield stress f_y and Young's Modulus E are

determined. The critical buckling load of a beam is given as

$$N_{cr} = \frac{\pi^2 \cdot E \cdot I}{l_k^2} \tag{1}$$

This calculation is done in lines 16 to 18 for the two principal axes of the cross section. A variable "h" acts as a helper, "M_PI" represents π. The procedure for calculating the design value of the buckling force N_{brd} is taken from Eurocode EN 1993-1-1:2005, paragraph 6.3.1.1, equation (6.47):

$$N_{b,Rd} = \frac{\chi \cdot A \cdot f_y}{\gamma_{M1}} = \chi \cdot A \cdot f_{yd} \tag{2}$$

Karamba assumes cross sections of class 1,2 or 3. This means that their most strained fibre yields before local buckling occurs. When you define a material Karamba further assumes that the given yield stress f_y is already reduced by the safety factor γ_{M1} which in case of steel is 1.0 in many European countries. Equation 6.49 of EC3 states that

$$\chi = \frac{1}{\Phi + \sqrt{\Phi^2 - \overline{\lambda}^2}} \leq 1.0 \tag{3}$$

The equivalent code can be found in lines 26 and 27. The values for $\overline{\lambda}$ and Φ are calculated according to the formulas of EC3 section 6.3.1.2:

$$\Phi = 0.5 \cdot [1 + \alpha \cdot (\overline{\lambda} - 0.2) + \overline{\lambda}^2] \tag{4}$$

$$\overline{\lambda} = \sqrt{\frac{A \cdot f_y}{N_{cr}}} \tag{5}$$

In Karamba $\overline{\lambda}$ gets calculated for the principal directions of the cross section (see lines 19 to 21). Φ_y and Φ_z for the principal directions get calculated in lines 22 and 23. The imperfection factors α_y ("alpha_y") and α_y ("alpha_z") are taken from the cross section tables. When using cross sections created within a GH-definition then $\alpha_y = \alpha_z = 0.3$ is assumed. The larger of Φ_y and Φ_z and the corresponding $\overline{\lambda}$ (see lines 24 and 25) are selected for calculating χ in lines 26 and 27.

In lines 29 to 35 the resisting moment and forces of the cross section are calculated. In case of elastic design the elastic resistance moments W_y and

W_z come into play, otherwise their plastic counterparts $W_{y,pl}$ and $W_{z,pl}$. A_y and A_z are the equivalent shear areas in y- and z-direction.

Inside the loop which comprises lines 45 to 75 the cross section resistance is compared to the cross section forces. This is done for the two endpoints of index zero and one. In order to assess the effect of combined loads the superposition formula in line 58 or 60 is used for axial stress. If the normal force is tensile ($N > 0$) then N is compared to the plastic resisting force N_{rd} (see line 29), if compressive then $N_{b,rd}$ enters the picture. These formulas correspond to that given in equation 6.41 of EC3. The superposition of the effect of shear in y- and z-direction and torsional moments is considered in line 63 along the same lines as for the axial stresses. The fact that shear and normal stress are considered separately constitutes a simplification. The larger of the utilization numbers of shear and normal stress is taken as the resulting cross section utilization (see line 66).

References

[1] J. H. Argyris, L. Tenek, and L. Olofsson. Tric: a simple but sophisticated 3-node triangular element based on 6 rigid.body and 12 straining modes for fast computational simulations of arbitrary isotropic and laminated composite shells. *Comput. Methods Appl. Mech. Engrg.*, 145:11–85, 1997.

[2] J.H. Argyris, M. Papadrakakis, C. Apostolopoulou, and S. Koutsourelakis. The tric shell element: theoretical and numerical investigation. *Comput. Methods Appl. Mech. Engrg.*, 182:217–245, 2000.

[3] H. Moldenhauer. Die visualisierung des kraftflusses in stahlbaukonstruktionen. *Stahlbau, Ernst & Sohn Verlag für Architektur und technische Wissenschaften GmbH & Co. KG, Berlin*, 81:32–40, 2012.